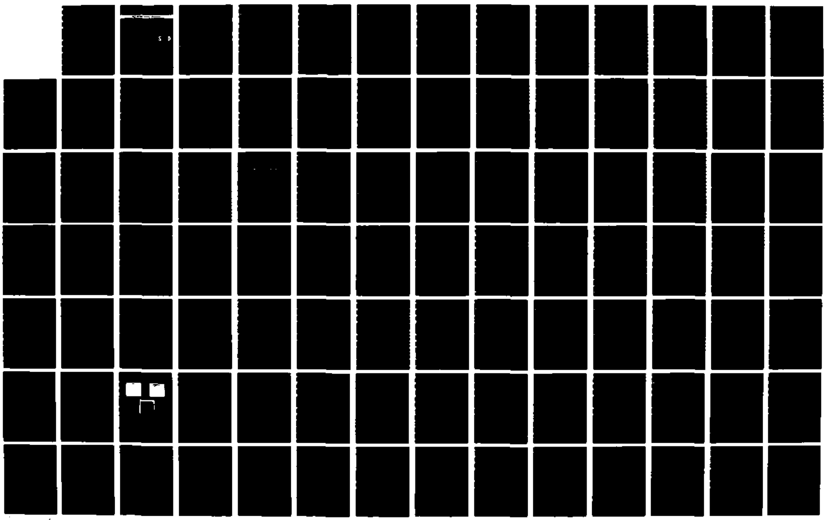
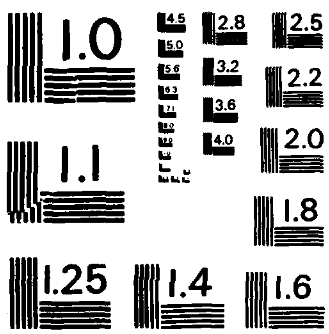
AD-A166 044 TRACK PARAMETER EXTRACTION USING MULTIPATH DELAY AND DOPPLER INFORMATION(U) SYSTEMS CONTROL TECHNOLOGY INC PALO ALTO CA K LASHKARI ET AL FEB 86 5517 1/1

UNCLASSIFIED N00014-84-C-0408 F/G 17/1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

(12)

SYSTEMS CONTROL TECHNOLOGY, INC.
1801 PAGE MILL RD. □ P.O. BOX 10180 □ PALO ALTO, CALIFORNIA 94303 □ (415) 494-2233

TRACK PARAMETER EXTRACTION USING MULTIPATH DELAY AND DOPPLER INFORMATION

FINAL REPORT 5517

Contract No. N00014-84-C-0408

FEBRUARY 1986

Prepared for:

Office of Naval Research
Department of the Navy
800 N. Quincy Street
Arlington, VA 22217-5000

Prepared by:

Khosrow Lashkari, Project Leader

Benjamin Friedlander

Jonathan Abel

Approved by:

Yair Barniv
Manager
Adaptive Systems Department

TABLE OF CONTENTS

SUMMARY

This final report summarizes the work performed under contract No. N00014-84-C-0408 to Office of Naval Research and is accompanied by the software listings of the developed code.

The goal of the multipath project is to develop techniques to make use of the multipath relections of acoustic signals travelling in the ocean to localize and track radiating sources. There have been two main areas of study under this contract. First area concentrates on the detection and estimation of the multipath (or differential) delays in the received signals; the second is the development of localization and tracking algorithms.

Previous work on the multipath project included the development of differential delay estimation algorithms as well as theoretical studies resulting in bounds on the variance of estimating multipath delay. Bounds were also placed on the accuracy of estimating various so-called track parameters -- speed, depth, and range at closest approach of the target to the sensor array.

Under the present contract, software was developed to estimate multipath delay using real or simulated data as input. Also under the present contract, tracking algorithms were developed. More specifically the following are accomplishments of this phase.

1. A real data processing system was developed for the purpose of estimating mutipath delay as a function of time. This system was composed of a correlogram generator and a line tracker. A correlogram is a picture whose rows consist of normalized correlation functions at different time instances. The correlation functions peak at the value of the differential delay and align to form tracks in the correlogram. The line tracker pulls the differential delay curves out of the correlogram. The correlogram generator produces SCOT, ML-, PHAT- and un-normalized correlograms. The line tracker uses the ADEC line tracking algorithm. This software was developed on the AP120B array processor on VAX 11/780 for high speed

computation.

2. Two types of track parameter estimation methods were developed. The first makes use of the measured value and functional form of the delay curve at various points in time. The other fits an estimated delay curve to the measured delay curve, and estimates the track parameters as those giving the best fit to the data. Target is assumed to travel along a constant-depth, constant-velocity straight-line course, and the parameters of the target's path -- depth, velocity, and closest point of approach -- are estimated, thereby localizing the target. The tracking algorithms were implemented in FORTRAN 77 and evaluated using the real data provided by ONR. The results are encouraging in that these methods allow tracking of real underwater targets.

TRACK PARAMETER EXTRACTION USING MULTIPATH DELAY AND DOPPLER INFORMATION

0. INTRODUCTION

Acoustic waves propagating in the ocean undergo various reflections and refractions [1]. Because of this multipath propagation, acoustic surveillance systems receive several different delayed and doppler shifted versions of the source's signal. The relative delays and doppler shifts contain valuable information which can help localize and track a target [2]. Consider, for example, a single multipath reflection depicted in Figure 0.1. The time delay between the signals propagating in the direct and reflected paths is a function of the target/sensor range and target and sensor depths. Similarly, the doppler shifts of the direct and multipath signals are functions of the velocity and range of the target. Therefore, a history of the delay and doppler shifts can lead to knowledge of the so-called track parameters -- speed, range of closest approach, and depth.

In this report, techniques for extracting track parameters from delay and doppler information are presented. Bounds are placed on the accuracy of these estimates as well as the accuracy of the delay and doppler measurements. Cases with one target, one and two sensors with and without multipath are treated in detail.

The structure of this report is as follows: Chapter one details the geometry and signal equations for a single sensor and target. This chapter also discusses the measurement of the delay and doppler shifts from the raw data. Chapter two presents the delay and doppler formulas for a single sensor and single target in the presence of a surface reflection. The accuracy of track parameter estimators using delay and doppler information is also discussed in Chapter two. Chapters three and four develop methods for extracting the track parameters from delay and doppler information for several sensor geometries; and chapter five contains some concluding remarks.

There are four appendices: appendix one discusses the effect

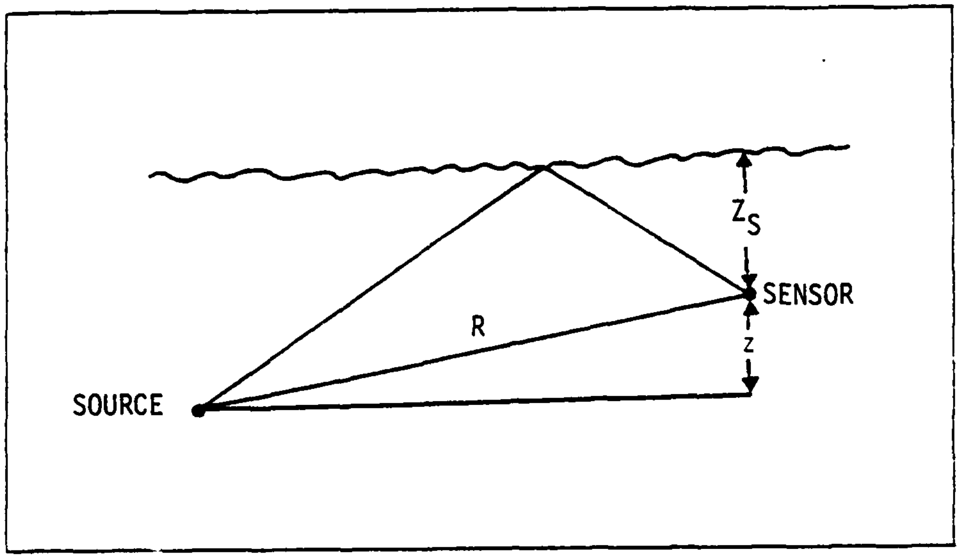


Figure 0.1 Illustration of multipath due to a surface bounce for the case of a single source and sensor. $D = (1/C)\ [(R^2+4z_S^2+4z_S z)^{1/2}-R]$ = multipath delay. C is the velocity of sound in water.

differential doppler has on the measurement of differential delay. Appendix two presents an example of track parameter estimators applied to a real data set. Appendix three discusses the accuracy of depth estimation using multipath delay. Finally, appendix four contains software listings.

1. MULTIPATH PARAMETERS FROM RAW DATA

In this chapter, the extraction of delay and doppler information from raw data is discussed. We consider here the case of a single multipath reflection (Fig. 1.1). The signal at the sensor will be roughly given by

$$y(t) = x(\alpha_d(t)\cdot t - D_d(t)) + g(t)x(\alpha_m(t)\cdot t_m - D\ (t)) + e(t) \qquad (1.1)$$

where

$y(t)$	is the signal at the sensor
$\alpha_d(t)$	is the doppler shift in the direct signal
$\alpha_m(t)$	is the doppler shift in the multipath signal
$D_d(t)$	is the delay in the direct signal
$D_m(t)$	is the delay in the multipath signal
$g(t)$	is the gain of the reflected signal
$e(t)$	is the sensor noise
$x(t)$	is the target signal

It is assumed that e is uncorrelated with x; the doppler shifts are small: α_d, $\alpha_m \sim 1$; and the delays, doppler shifts and gains are slowly changing with time: $|\frac{\partial D}{\partial t}|$, $|\frac{\partial \alpha}{\partial t}|$, $|\frac{\partial g}{\partial t}| << 1$; and $0 \leq |g| < 1$.

1.1 DELAY

Assuming $\alpha_d = \alpha_m = 1$, the Cramer-Rao lower bound on the variance of the minimum variance estimator of the differential delay, $D = |D_d - D_m|$, is reasonably small for signals with a large bandwidth-delay product [3]. When x

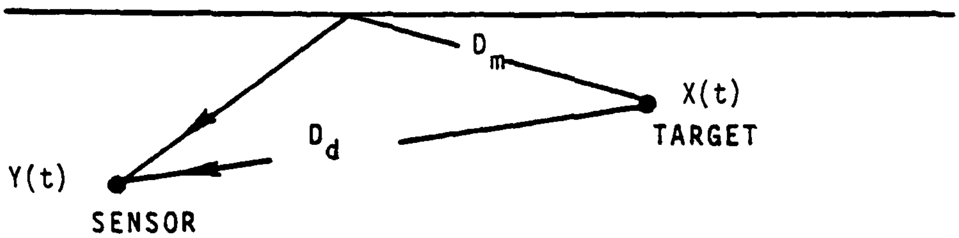


Figure 1.1 The received sensor signal in the presence of a multipath reflection.

and e are uncorrelated lowpass white noise processes with bandwidth w, the variance in estimating D from the signal $y(t) = x(t) + g \cdot x(t-D) + e(t)$ is bounded by,

$$\mathrm{Var}(D) > \frac{3\pi}{Tw^3 g^2} \frac{1}{SNR^2} \quad , \quad SNR \ll 1, \; WD \gg 1 \tag{1.2a}$$

$$\mathrm{VAR}(D) > \frac{3\pi}{w^3 g^2}(1-g^2) \quad , \quad SNR \gg 1, \; WD \gg 1 \tag{1.2b}$$

where the signal-to-noise ratio (SNR) is the ratio of the power contained in the signal and noise. The maximum likelihood (ML) estimator achieves this bound, but involves a computationally intensive minimization of a complicated cost function [2].

Maximizing a simplified version of the ML cost function results in the autocorrelation method of multipath delay estimation: the delay is chosen as the value of τ which maximizes $|R_{YY}(\tau)|$, the absolute value of the autocorrelation function [2]. The autocorrelation function is relatively easy to compute, and is commonly used for delay estimation [4].

The variance and bias in estimating delay using the autocorrelation method based on data over a time period, T are given in [2] by

$$\mathrm{Var}\{D^*\} = \frac{3\pi}{Tw^3 g^2}\left(\frac{1}{SNR}\right)^2 \; , \; SNR \ll 1, \; WD \gg 1 \tag{1.3a}$$

$$\mathrm{BIAS} = < \frac{3}{w^2 D} \sqrt{\frac{1}{4} + \left(\frac{1}{g} + g + \frac{S_e}{gS_x}\right)^2} \tag{1.3b}$$

Note that for low SNR, the ML estimator and the autocorrelation estimator have essentially the same variance. Therefore, for sufficiently large bandwidth, low SNR signals, as is usually the case in underwater acoustic surveillance, the correlation estimator will perform almost optimally.

Tracking Differential Delay D(t)

For $\dot{D}(t) << 1$, and $\alpha_m = \alpha_d = 1$, the autocorrelation function and power spectrum of y(t) are given by

$$R_{YY}(\tau) = (1+g^2)R_{XX}(\tau) + gR_{XX}(\tau-D) + gR_{XX}(\tau+D) + R_{ee}(\tau) \tag{1.4a}$$

$$S_{YY}(\omega) = (1 + g^2 + 2g\cos\omega D)S_{XX}(\omega) + S_{ee}(\omega) \tag{1.4b}$$

From (1.4), it is easily seen that when the width of $R_{XX}(\tau) << 1$, peaks of $R_{YY}(\tau)$ will occur at the value of the multipath delay as well as at zero delay.

When the SNR is low, and D(t) is slowly changing with time, D(t) is most often found using a correlogram -- a 2-D gray level image whose i-th row is the correlation function taken at t_i, $E(Y(t_i)Y(t_i+\tau))$ See Figure A3.1 for example. Peaks in the correlation functions comprising a correlogram will merge to form lines tracing the history of D(t).

Doppler Effects

Since the direct and multipath signals are arriving from different directions, a moving target will cause a relative doppler shift between the two signals. The autocorrelation of the sensor signal will become

$$\begin{aligned} R_{YY}(t,\tau) = {} & R_{XX}[(t-\tau)\alpha_d] + g^2R_{XX}[(t-\tau)\alpha_m] \\ & + gR_{XX}[\alpha_d t, \alpha_m\tau - D] + gR_{XX}[\alpha_d t, \alpha_m\tau + D] + R_{ee}(t-\tau)] \end{aligned} \tag{1.5}$$

where

$$R_{XX}[a,b] = E[X(a)X(b)] \text{ , and } R_{XX}[\tau] = E[X(t)X(t+\tau)]$$

As shown in Appendix 1, the secondary peaks in $R_{YY}(\tau)$ which are used to estimate the delay are reduced as the differential doppler, $\alpha_d - \alpha_m$, increases. That is, the direct and multipath signals become decorrelated as a

result of target motion.

The differential doppler is greatest when the difference in the directions of arrival of the direct and multipath signals is greatest. This occurs at CPA, the point of closest approach of the target to the sensor array. The fading of the correlogram lines near CPA in Figure A2.1 is in part due to this differential doppler effect.

1.2 DOPPLER

Assuming $D_d = D_m = 0$, substituting $t' = \log t$ into (1.1) gives

$$Y(t') = X(t'+\log\alpha_d) + X(t'+\log\alpha_m) + e(t') \tag{1.6}$$

Note that since e(t) is white gaussian noise, so is e(t'). Therefore, estimating differential doppler in the absence of delay is equivalent to estimating differential delay in the absence of doppler.

The ML estimator of doppler shift achieves the C-R bound of

$$\mathrm{Var}(\log\frac{\alpha_d}{\alpha_m}) = \frac{3\pi}{W'^3 g^2}\frac{1}{SNR^2}, \quad SNR \ll 1 \tag{1.7a}$$

$$\mathrm{Var}(\log\frac{\alpha_d}{\alpha_m}) = \frac{3\pi}{W'^3 g^2}(1-g^2), \quad SNR \gg 1 \tag{1.7b}$$

where W' is the bandwidth of the log sampled signal, X(t'). The autocorrelation method of determining differential delay estimation, looking for the peak of $E(Y(t')Y(t'+\log\alpha))$, i.e., the peak of $E(Y(t)Y(\alpha t))$, has variance

$$\mathrm{Var}(\log\alpha) = \frac{3\pi}{W'^3 g_0^2}\frac{1}{SNR^2}, \quad SNR \ll 1 \tag{1.8}$$

where it is assumed that $\log\alpha \cdot W' \ll 1$.

<u>Tracking $\alpha(t)$</u>

The doppler ratio, $\alpha = \frac{\alpha_m}{\alpha_d}$ is often seen using dopplergram. Analogous

to a correlogram, a dopplergram is a 2-D image whose i-th row is $E(Y(t_i)Y(\alpha t_i))$. Lines in the dopplergram show the time history of $\alpha(t)$.

Delay Effects

As differential doppler decorrelated the multipath and direct linearly sampled signals, differential delay will decorrelate the multipath and direct log sampled signals:

$$E[Y(t)Y(\alpha^{*}t-D)] \approx gR_{XX}(D) < gR_{XX}(0) \qquad (1.9)$$

where α^{*} is the correct value of the doppler ratio. Differential delay is greatest at CPA. Consequently, one would expect fading of differential doppler lines around CPA.

Absolute Doppler

As a final note, absolute doppler information can be obtained by following the motion of narrow band components in the spectrum of the received signal. A 2-D image made of spectra taken at different times, a spectrogram, is often used for this purpose.

2. DELAY AND DOPPLER EQUATIONS, AND ACCURACY ANALYSIS

This chapter develops basic delay and doppler formulas used in finding track parameters for measurements of differential delay and doppler; bounds are placed on the variance of track parameter estimators using these measurements. It is assumed here that the ocean is a homogeneous medium with a reflecting boundary at the surface ($z = 0$).

2.1 DELAY AND DOPPLER EQUATIONS

With a sensor at (X_s, Y_s, Z_s) and a target at (Vt, Y_T, Z_T), the direct, multipath, and differential delay between direct and multipath signals are given by (see Figs. 1.1, and 2.1):

$$D_d(t) = [(X_s-Vt)^2 + (Y_s-Y_T)^2 + (Z_s-Z_T)^2]^{1/2}/C \tag{2.1a}$$

$$D_m(t) = [(X_s-Vt)^2 + (Y_s-Y_T)^2 + (Z_s+Z_T)^2]^{1/2}/C \tag{2.1b}$$

$$D(t) \triangleq D_m(t) - D_d(t) \tag{2.1c}$$

where C is the speed of sound in water. The differential delay rate and differential delay curvature are:

$$\frac{\partial D}{\partial t} = \frac{V}{C^2}(Vt-X_s)\left[\frac{1}{D_m} - \frac{1}{D_d}\right] \tag{2.2a}$$

$$\frac{\partial^2 D}{\partial t^2} = \frac{V^2}{C^2}\left[\frac{1}{D_m} - \frac{1}{D_d}\right] + \frac{(Vt-X_s)^2}{C^4}\left[\frac{1}{D_d^3} - \frac{1}{D_m^3}\right] \tag{2.2b}$$

The doppler shift of a source frequency, f_s is given by

$$f = f_s\left(1 + \frac{V_r}{C}\right) = f_s \cdot \delta \tag{2.3}$$

where f is the received frequency, and V_r is the velocity of the target along a line connecting the target and sensor, and δ is the doppler shift. The direct, multipath, and differential doppler shift between the direct and

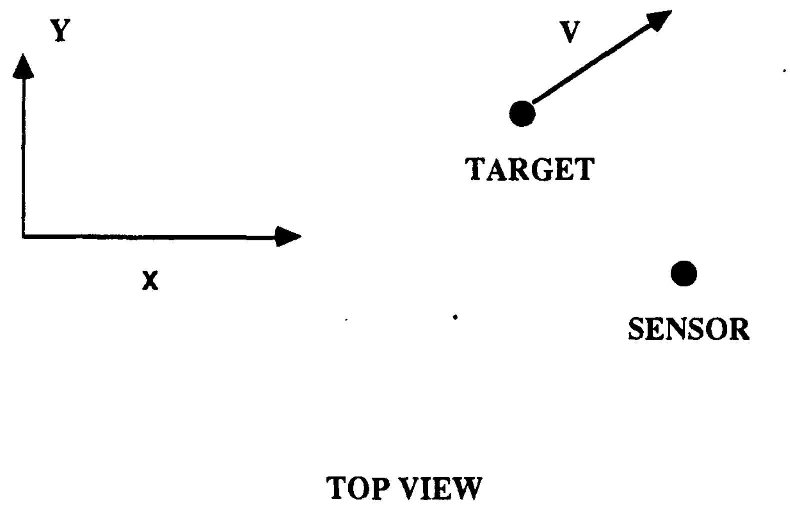


Figure 2.1 View from above the ocean of the target and sensor. The target is located at $(X_T\text{-}Vt,\ Y_T,\ Z_T)$ and the sensor is located at, $(X_S,\ Y_S,\ Z_S)$.

multipath signals are given by

$$\delta_d(t) = 1 + \frac{V(Vt-X_s)}{C^2 D_d} = 1 + \frac{\partial D_d(t)}{\partial t} \tag{2.4a}$$

$$\delta_m(t) = 1 + \frac{V(Vt-X_s)}{C^2 D_m} = 1 + \frac{\partial D_m(t)}{\partial t} \tag{2.4b}$$

$$\delta(t) = \delta_m(t) - \delta_d(t) = \frac{\partial D(t)}{\partial t} \tag{2.4c}$$

The differential delay between direct paths from the target to two sensors is given by

$$D_{12} = D_{d1} - D_{d2} , \qquad (2.5a)$$

where D_{d1} and D_{d2} are the delays between the target and the individual sensors. The differential doppler between the direct paths of the two sensors is again given by the differential delay rate:

$$\delta_{12} = \delta_{d1} - \delta_{d2} = \frac{\partial D_{12}}{\partial t} \qquad (2.5a)$$

$$\frac{\partial \delta_{12}}{\partial t} = \frac{\partial^2 D_{12}}{\partial t^2} \tag{2.5b}$$

2.2 ACCURACY ANALYSIS

It is of interest to know how well the track parameters can be estimated from multipath delay information. The C-R bound can be used in conjunction with the above formulas to put a lower bound on the variance of any unbiased estimators of the track parameters.

C-R BOUND CALCULATION

It is assumed that data is available during the time period (-T,T). Over this time period, the data are nonstationary. Since the C-R bound can be used

only with stationay data, for purposes of determining a bound on the variance of the track parameter estimates, it is assumed that estimates of track parameters are made from sub-intervals of time of length Δt (over which the data are assumed stationary) and linearly combined to form a single estimate of the track parameters. In this case the variance in estimating a track parameter, θ is given by,

$$\sigma_{\hat{\theta}}^2 = \left(\sum_{i=-N}^{N} \frac{1}{\sigma^2(i)} \right)^{-1} \tag{2.6a}$$

and $\hat{\theta}$ is given by,

$$\hat{\theta} = \left(\sum_{i=-N}^{N} \frac{\hat{\theta}(i)}{\sigma_\theta^2(i)} \right) / \left(\sum_{i=-N}^{N} \sigma_\theta^2(i) \right) \tag{2.6b}$$

where the variances, $\sigma_\theta^2(i)$ are bounded below by the C-R bound,

$$\sigma_\theta^2(i) > \frac{1}{I_\theta(i)} \tag{2.7}$$

where, $I_\theta(i)$ is the Fisher information matrix element corresponding to the parameter θ at time t_i .

The variance in estimating θ from time delay measurements over the interval (-T,T) is therefore bounded by,

$$\sigma_\theta^2 > \frac{1}{\sum\limits_{i=-N}^{N} I_\theta(i)} \tag{2.8}$$

which can be approximated by,

$$\sigma_\theta^2 \gtrsim \left[\int_{-\tau}^{\tau} I_\theta(t)dt \right]^{-1} \tag{2.9}$$

for large N. The Fisher information matrix element, $I_\theta(i)$, can be evaluated exactly using methods described in Appendix 3. These expressions are algebraically complicated, and difficult to manipulate. If it is assumed that all track parameters but θ are known, then $I_\theta(t)$ is given simply by,

$$I_\theta(t) = \left(\frac{\partial D}{\partial \theta}\right)^2 I_D(t) \tag{2.10}$$

where $I_D(t)$ is the Fisher information matrix element for the delay, evaluated in section 1. Note that the Fisher information above is larger than that calculated assuming the other track parameters are unknown. Therefore, the resulting bounds on the variance of the track parameter estimates will not be tight.

Using (2.1), (2.10), and (2.9), a lower bound can be placed on the variance of track parameter estimates,

$$\sigma_\theta^2 > \left[\int_{-T}^{T} \left(\frac{\partial D(t)}{\partial \theta}\right)^2 I_D(t)dt\right]^{-1} \tag{2.21}$$

where, from (1.2a),

$$I_D(t) \sim w^3 g^2 SNR^2/3$$

for low SNR broadband signals.

DEPTH ESTIMATE VARIANCE

Assuming other track parameters are known, the Fisher information of the depth estimate based on delay measurements is given by (2.10) as,

$$I_{Z_T}(t) = \frac{1}{C^4}\left[\frac{(Z_T-Z_S)^2}{D_d^2} + \frac{(Z_T+Z_S)^2}{D_m^2} + 2\,\frac{Z_s^2 + Z_T^2}{D_d D_m}\right] I_D(t) \ , \tag{2.12}$$

Using (2.21) and integrating,

$$\frac{1}{\sigma_{Z_T}^2} < \left(\frac{w^3 g^2 SNR^2}{3C^3 V}\right) \cdot \left\{\frac{(Z_T-Z_S)^2}{\alpha_d} \tan^{-1}\frac{\tau}{\alpha_d} + \frac{(Z_T+Z_S)^2}{\alpha_m} \tan^{-1}\frac{\tau}{\alpha_m}\right. \tag{2.13}$$

$$\left. + 2\,\frac{Z_S^2 - Z_T^2}{\alpha_d} \tan^{-1}\frac{\tau}{\alpha_d}\right\}\Bigg|_{\tau=X_T-VT}^{\tau=X_T+VT}$$

where

$$\alpha_d = (Y_T^2 + (Z_T - Z_S)^2)^{1/2}$$

$$\alpha_m = (Y_T^2 + (Z_T + Z_S)^2)^{1/2}$$

$$\alpha_+ = (Y_T^2 + Z_T^2 + Z_S^2)^{1/2}$$

In the case of,

$$X_T \pm VT << \alpha_d, \alpha_m, \alpha_+ \tag{2.14}$$

and,

$$Y_T^2 >> Z_S^2, Z_T^2 \tag{2.15}$$

$\frac{1}{\sigma_{Z_T}^2}$ reduces to,

$$\frac{1}{\sigma_{Z_T}^2} \leq \left(\frac{w^3 q^2 SNR^2}{3C^4}\right) \cdot \left(\frac{4Z_S^2}{Y_T^2}\right) \cdot 2T \tag{2.16}$$

Notice that this expression is proportional to T, i.e. the more data available, the lower the variance. Also notice, as the depth of the sensor increases relative to the range at CPA, the minimum variance is decreased.

When (2.15) and,

$$|X \pm VT| >> \alpha_d, \alpha_m, \alpha_+ \tag{2.17}$$

hold, i.e. most all of the delay curve is available,

$$\frac{1}{\sigma_{Z_T}^2} \leq \left(\frac{w^3 q^2 SNR^2}{3C^4}\right)\left(\frac{\pi}{V}\right)\left(\frac{4Z_S^2}{Y_T}\right) \tag{2.18}$$

In this case, the variance in estimating the target depth is dependent on the sensor depth, the deeper the sensor, the better the depth estimate; and the target velocity and CPA range, closer slower moving targets give better depth estimates.

Y OFFSET

With (2.11) and (2.1), $\frac{1}{\sigma_{Y_T}^2}$ is, in the case of other track parameters known, given by,

$$\frac{1}{\sigma_{Y_T}^2} \leq \left(\frac{w^3 g^2 SNR^2}{3V}\right)\left(\frac{Y_T^2}{C^4}\right)\left\{\frac{1}{\alpha_d}\tan^{-1}\frac{\tau}{\alpha_d} + \frac{1}{\alpha_m}\tan^{-1}\frac{\tau}{\alpha_m} - \frac{2}{\alpha_+}\tan^{-1}\frac{\tau}{\alpha_+}\right\}\Bigg|_{\tau=X_T-VT}^{\tau=X_+ +VT} \tag{2.19}$$

If only data near CPA is avaliable, i.e. (2.14) applies, then under (2.15), $\frac{1}{\sigma_{Y_T}2}$ reduces to,

$$\frac{1}{\sigma_{Y_T}2} \leq \left(\frac{w^3 g^2 SNR^2}{3C^4}\right)(4T)\left(\frac{Z_S^2 Z_T^2}{Y_T^4}\right) \tag{2.20}$$

In other words, the $\sigma_{Y_T}^2$ is reduced for increasing Z_S, Z_T, and T and decreasing target CPA range.

When a lot of data are available, i.e. when (2.17) is applicable, $\sigma_{Y_T}^2$ becomes,

$$\frac{1}{\sigma_{Y_T}^2} \leq \frac{w^3 g^2 SNR^2}{3c^4} \cdot \frac{8}{V} \cdot \frac{Z_s^2 Z_T^2}{Y_T^3} \tag{2.21}$$

In this case, more data no longer significantly improves the estimate. Note here that the slower the target moves, the broader the delay curve, and the better the estimate.

X OFFSET

Using (2.10) and (2.11) the variance of the X offset can be bounded by,

$$\frac{1}{\sigma^2_{X_T}} \le \frac{w^3 g^2 SNR^2}{3c^4 V}\left\{-\alpha_d \tan^{-1}\frac{\tau}{\alpha_d} - \alpha_m \tan^{-1}\frac{\tau}{\alpha_m} + 2\alpha_+ \tan^{-1}\frac{\tau}{\alpha_+}\right\}\Bigg|_{\tau=X_T-VT}^{\tau=X_T+V_T} \qquad (2.22)$$

Under the conditions (2.14) and (2.15),

$$\frac{1}{\sigma^2_{X_T}} \le \frac{w^3 g^2 SNR^2}{2c^4} \cdot \frac{4}{3} V^2 T^3 \left(\frac{Z_s^2 Z_T^2}{Y_T^6}\right) \qquad (2.23)$$

Note here the heavy dependence of $\sigma^2_{X_T}$ on v,T, and Y_T : the more data available near CPA, and the more peaked the delay curve is, the easier it is to estimate $\sigma^2_{X_T}$.

When (2.27) applies,

$$\frac{1}{\sigma^2_{X_T}} \le \frac{w^3 g^2 SNR^2}{3c^4} \cdot \frac{\pi}{V} \frac{Z_s^2 Z_1^2}{Y_T^3} \qquad (2.24)$$

In this case, the estimate variance is no longer as dependent on the shape of the delay curve or the amount of data available.

VELOCITY

Again, using (2.10) and (2.11), a bound on the variance of the velocity estimate can be found as,

$$\frac{1}{\sigma^2_V} \le \frac{w^3 g^2 SNR^2}{3c^4} \cdot \frac{1}{V^3}\Big[\left(\alpha_d^3 \tan^{-1}\frac{\tau}{\alpha_d} + \alpha_m^3 \tan^{-1}\frac{\tau}{\alpha_m} - 2\alpha_+^3 \tan^{-1}\frac{\tau}{\alpha_+}\right)$$

$$+ 2_{X_T}\left(\frac{\alpha_d^2}{2} \log(\alpha_d^2 + \tau^2) + \frac{\alpha_m^2}{2} \log(\alpha_m^2 + c^2) - \alpha_+^2 \log(\alpha_+^2 + \tau^2)\right)$$

$$+ X^2\left(-\alpha_d \tan^{-1}\frac{\tau}{\alpha_d} - \alpha_m \tan^{-1}\frac{\tau}{\alpha_m} + 2_{\alpha_+} \tan^{-1}\frac{\tau}{\alpha_+}\right)\Big]\Bigg|_{\tau=X_T-VT}^{\tau=X_T+VT} \qquad (2.25)$$

When (2.14) and (2.15) are satisfied,

$$\frac{1}{\sigma^2_V} \sim \frac{w^3 g^2 SNR^2}{3c^4} \cdot (2V^5 T^5 + X_T V^4 T^4 + \frac{2}{3} X_T^2 V^3 T^3)\ 2\left(\frac{Z_s^2 Z_T^2}{Y_T^6}\right) \qquad (2.26)$$

Again, the estimate variance is sensitive to V, T, Y_T ; lower variance estimates are obtained from data near CPA where the delay curve is more peaked.

If instead of (2.14), (2.15) is applicable, then

$$\frac{1}{\sigma_V^2} \sim \frac{2\pi W^3 g^2 SNR^2}{3c^4} \cdot \frac{(Z_S^2 Z_T^2)}{V^3 Y_T} \qquad (2.27)$$

Notice, in this case, the variance of the velocity estimate increases with the "sharpness" of the delay, and not the "flatness" as was the case in (2.26).

SUMMARY

In summary, provided the delay estimates are good, i.e. the banwidth of the signal is large enough, and provided the aperture size of the array is large, i.e. Z_S is large enough compared to the CPA range of the target, reasonably small lower bounds on the variance of the track parameter estimates are obtained.

Tighter bounds on the track parameter estimates as well as bounds for estimates based on inter-sensor and multipath delay information from two sensor arrays are presented in appendix 3.

3. POINT METHOD OF TRACK PARAMETER EXTRACTION

Techniques for extracting track parameters from correlograms, dopplergrams, and spectrograms are presented in this chapter. The parameters of interest are V, the velocity of the target; z_T, the depth of the target; R_{CPA}, the radius of closest approach to the sensor array; and θ , the bearing angle relative to the array.

3.1 SINGLE SENSOR

The case of a single sensor at $(0,0, Z_S)$ and a single target at (Vt,Y_T, Z_T) with one surface reflection is considered first. Figure 3.1 shows the correlogram for this case. Differential delay rate, estimated from the dopplergram is plotted in Figure 3.2. These measurements lead to estimates relating R_{CPA}, V, and Z_T . Measurements of the doppler shift of spectral lines give velocity as well as R_{CPA} estimates.

Delay Measurements

When the target is far away from the sensor, the differential delay becomes less dependent on the relative y-axis positions of the sensor and target. As $|t| \to \infty$, a relationship between z_T and V can be developed.

From (2.1) for $|t| >> 1$,

$$D_d \sim [(Vt)^2 + Y_T^2]^{1/2}[1 + \frac{1}{2}\frac{(Z_S-Z_T)^2}{(Vt)^2+Y_T^2}]/C ,$$

$$D_m \sim [(Vt)^2 + Y_T^2]^{1/2}[1 + \frac{1}{2}\frac{(Z_S+Z_T)^2}{(Vt)^2+Y_T^2}]/C ,$$

$$D = D_m - D_d \sim \frac{2Z_sZ_T}{C[(Vt)^2+Y_T^2]^{1/2}}$$

Since for $|t| >> 1$, $(Vt)^2 >> Y_T^2$,

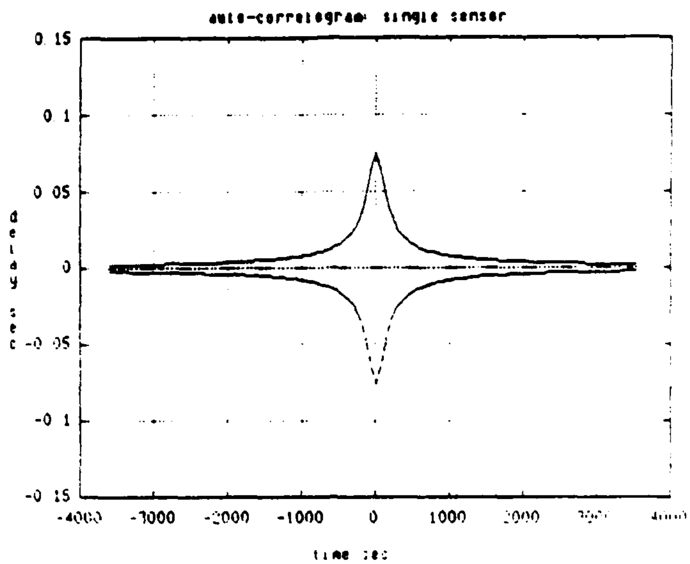


Figure 3.1 The autocorrelogram of a target at $(Vt, 0, Z_T)$ passing by a sensor at $(0, 0, Z_S)$ in the presence of a surface reflection.

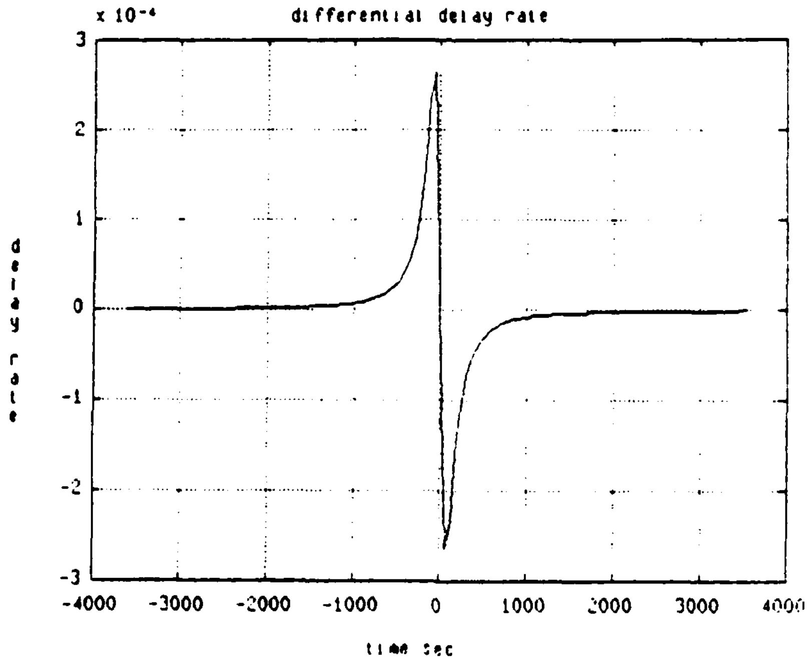


Figure 3.2 Differential delay rate (differential doppler between the direct and multipath signals for a target located at (Vt, D, Z_T) passing by a sensor at $(0, 0, t_s)$.

$$D \sim \frac{2Z_s Z_T}{C\,Vt}, \quad |t| >> 1 \tag{3.1}$$

Performing a least squares fit of ρ/t to the tails of the measured D(t) yields the relationship:

$$\frac{Z_T}{V} = \hat{\rho}\,\frac{C}{2Z_s}, \tag{3.2}$$

where $\hat{\rho} = \min_{\rho}[D(t) - \frac{\rho}{t}]^2$ for $t >> 1$.

A relationship between R_{CPA} and Z_T can be found by examining D(0), a readily available quantity. From (2.1) and, $R_{CPA} = D_d(0) = [Y_T^2 + (Z_s - Z_T)^2]^{1/2}$,

$$CD(0) = [R_{CPA}^2 + 4Z_s Z_T]^{1/2} - R_{CPA} \tag{3.3}$$

Solving (3.3) for R_{CPA},

$$R_{CPA} = \frac{2Z_s Z_T}{CD(0)} - \frac{CD(0)}{2} \tag{3.4}$$

Using a velocity estimated by other means, R_{CPA} and Z_T can be obtained using (3.2) and (3.4) with a few measurements of D(t).

Doppler Measurements

By following doppler shifts of lines in the spectrogram, V and R_{CPA} can both be estimated. Figure 3.3 shows the spectrogram of a constant velocity target emitting a single strong spectral line. Using (2.3) and (2.4),

$$f = f_s(1 + \frac{V_r}{C}) = f_s(1 + \frac{V(Vt - X_s)}{C^2 D_d}) \tag{3.5}$$

as $t \to -\infty$, $D_d \to D_m \frac{Vt - X_s}{C}$, and

$$f \to f_{max} \triangleq (1 + \frac{V}{C})f_s \tag{3.6}$$

Similarly, as $t \to +\infty$,

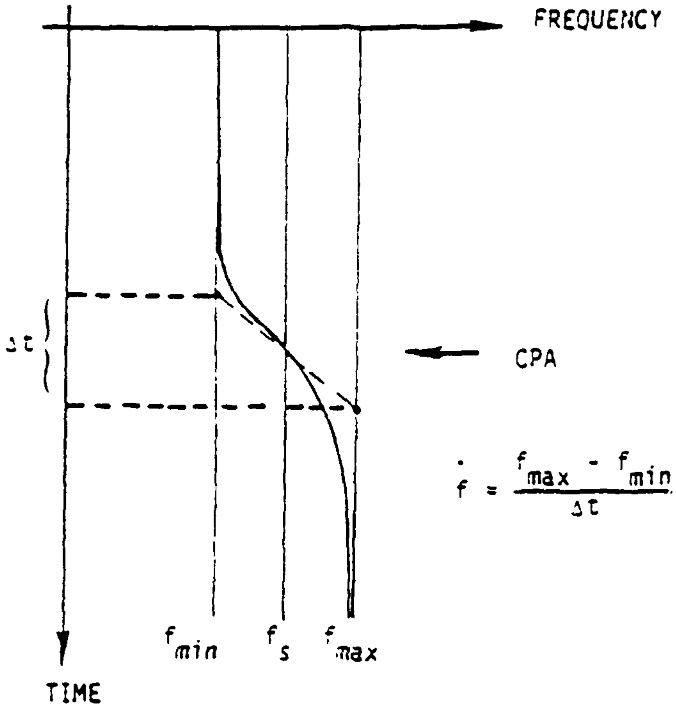


Figure 3.3 Doppler shift vs. time for a constant velocity source

$$f \rightarrow f_{min} \triangleq (1 - \frac{V}{C}) f_s \tag{3.6}$$

from (3.6) and (3.7) an expression for the velocity is obtained,

$$f_s = \frac{1}{2} (f_{max} - f_{min}) \tag{3.8}$$

and

$$V = C \frac{f_{max} - f_{min}}{f_{max} + f_{min}} \tag{3.9}$$

Range information can be found by examining $\frac{\partial f}{\partial t}$ at CPA:

$$\left.\frac{\partial f}{\partial t}\right|_{CPA} = \frac{f_s}{C} \left.\frac{\partial V_r}{\partial t}\right|_{CPA} \tag{3.10}$$

from (3.7)

$$\left.\frac{\partial f}{\partial t}\right|_{CPA} = f_s \left.\frac{\partial^2 D_d}{\partial t^2}\right|_{CPA} = \frac{f_s}{C} \frac{V^2}{R_{CPA}} \tag{3.11}$$

The time of maximum differential doppler is easily obtained from the dopplergram, and relates the direct and multipath CPA ranges. The maximum differential doppler time, t*, is given by (see (2.4)),

$$\ddot{D}(t) = \ddot{D}_m(t^*) - \ddot{D}_d(t^*) = 0 \tag{3.12}$$

Using (3.2)

$$D_m - D_d = \frac{V^2 t_*^2}{C^2} \left[\frac{D_m^3 - D_d^3}{D_m^2 D_d^2}\right]$$

$$D_m^2 D_d^2 = \frac{V^2 t_*^2}{C^2} (D^2 t + 3 D_m D_d)$$

Assuming $\frac{D^2}{D_m D_d} << 3$,

$$D_m D_d \sim 3 \frac{V^2 t_*^2}{C^2}$$

Therefore,

$$t_*^2 = \frac{C^2}{3V^2} D_m(t_*)D_d(t_*) \tag{3.13}$$

$$\delta(t_*) = \dot{D}(t_*) = \frac{D(t_*)}{3t_*} \tag{3.14}$$

Using (2.1), (2.2) and

$$R_d = Y_T2 + (Z_s - Z_T)^2$$

$$R_m = Y_T^2 + (Z_s + Z_T)^2 ,$$

the time of maximum differential delay is given by,

$$t_* = \frac{1}{4V} [R_d^2 + R_m^2 + (R_d^2 + R_m^2)^2 + 32R_d^2R_m^2)^{1/2}]^{1/2} \tag{3.15}$$

Summary

To summarize, from the correlogram and spectrogram, all track parameters can be estimated.

$$\hat{V} = C \frac{f_{max} - f_{min}}{f_{max} + f_{min}} \tag{3.16a}$$

$$\hat{R}_{CPA} = \frac{1}{2} C \frac{(f_{max} - f_{min})^2}{f_{max} + f_{min}} \left(\frac{\partial f}{\partial t} \bigg|_{t_{R_{CPA}}} \right)^{-1} \tag{3.16b}$$

$$\hat{\rho} = \underset{\rho}{\mathrm{Min}} [D(t) - \rho/t]^2 \tag{3.17a}$$

$$\hat{Z}_T = \hat{\rho}\ \hat{V} \cdot \frac{C}{2Z_s} \tag{3.17b}$$

$$\hat{R}_{CPA} = \frac{2Z_s\hat{Z}_T}{CD(0)} - \frac{CD(0)}{2} \tag{3.17c}$$

where f_{max}, f_{min}, $\left.\frac{\partial f}{\partial t}\right|_{t=RCPA}$, are measured from the spectrogram; $\hat{\rho}$ and D(0) are measured from the correlogram.

3.2 TWO SENSOR TRACK PARAMETER ESTIMATION

With the addition of a second sensor, the track parameter estimation capability of an array is greatly improved. First, V, Z_T and R_{CPA} estimates from each sensor can be combined to form estimates of lower variance. Second, measurements of intersensor delay and doppler lead to further estimates of V, Z_T, R_{CPA}, and possibly θ - the bearing angle relative to the array.

Two commonly used two-sensor arrays are considered here, the vertical array and the horizontal array. The arrays are shown in Figure 3.4; theoretical correlograms in the presence of multipath are shown in figures 3.5 and 3.7.

The Vertical Array

The vertical array depicted in Figure 3.4 has two sensors placed one above the other. This array is radially symmetric, and therefore can only estimate Z_T, V, and R_{CPA}. Separate estimates of these parameters can be made from multipath information at each sensor as well as inter-sensor delay and doppler shift data. Cross correlation and cross differential doppler are shown in Figure 3.5.

As can be seen in Figure 3.6, the delay between the direct and multipath propagation paths in the single sensor case is equivalent to the intersensor direct path delay in the 2-sensor vertical array. If the sensors of the vertical array are placed at

S1: $(0,0,Z_{s1})$

S2: $(0,0,Z_{s2})$

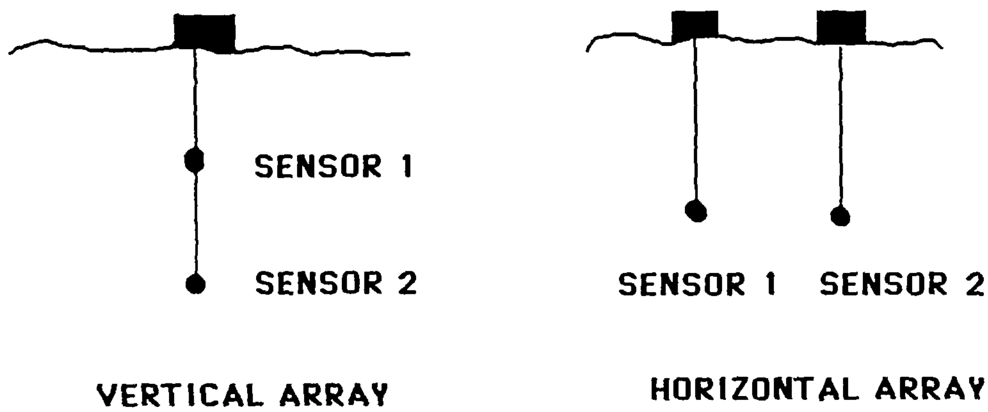


Figure 3.4 Vertical and horizontal 2-sensor arrays

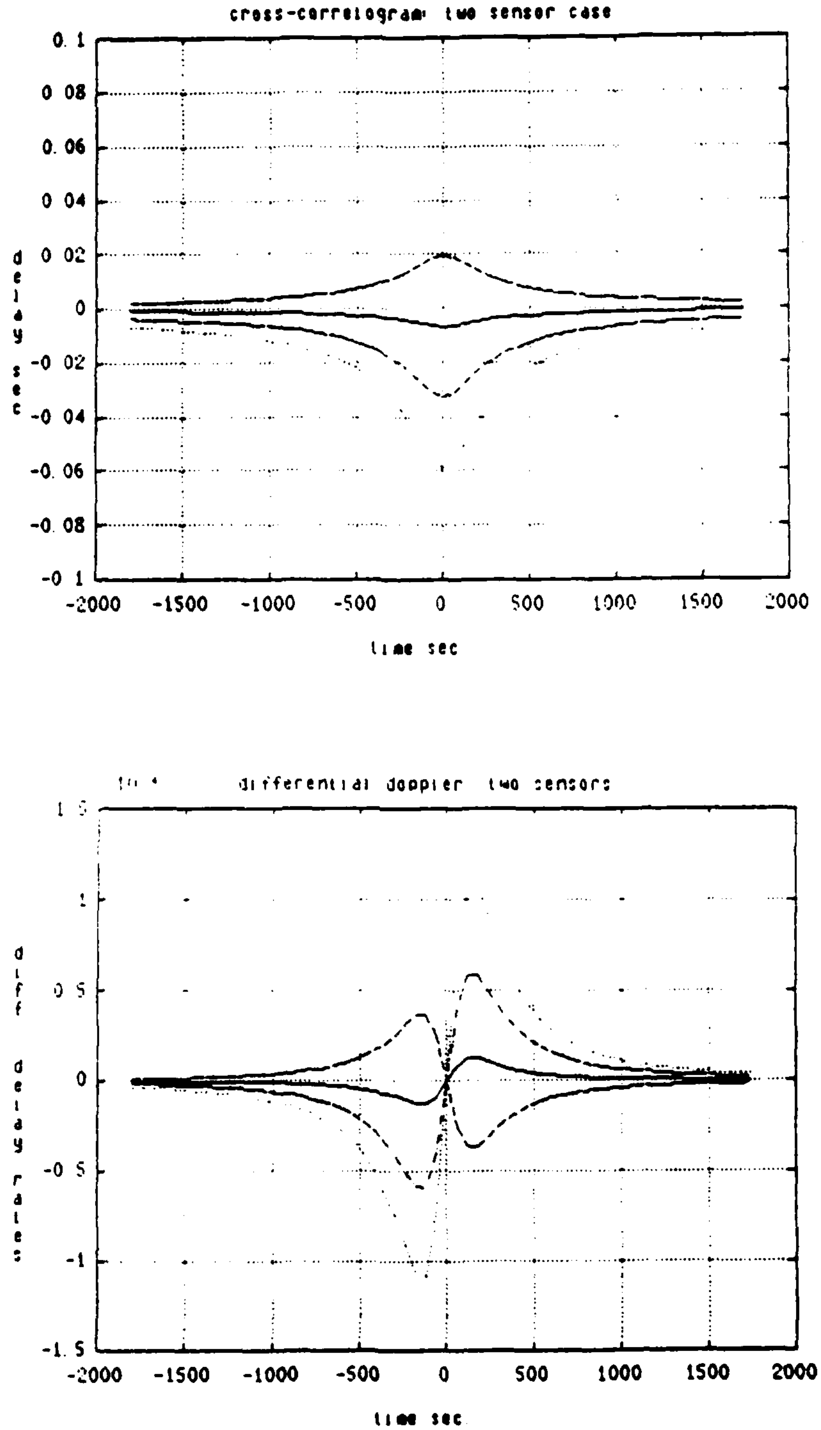


Figure 3.5 The cross-correlogram and differential doppler for a two sensor vertical array in the presence of a multipath reflection.

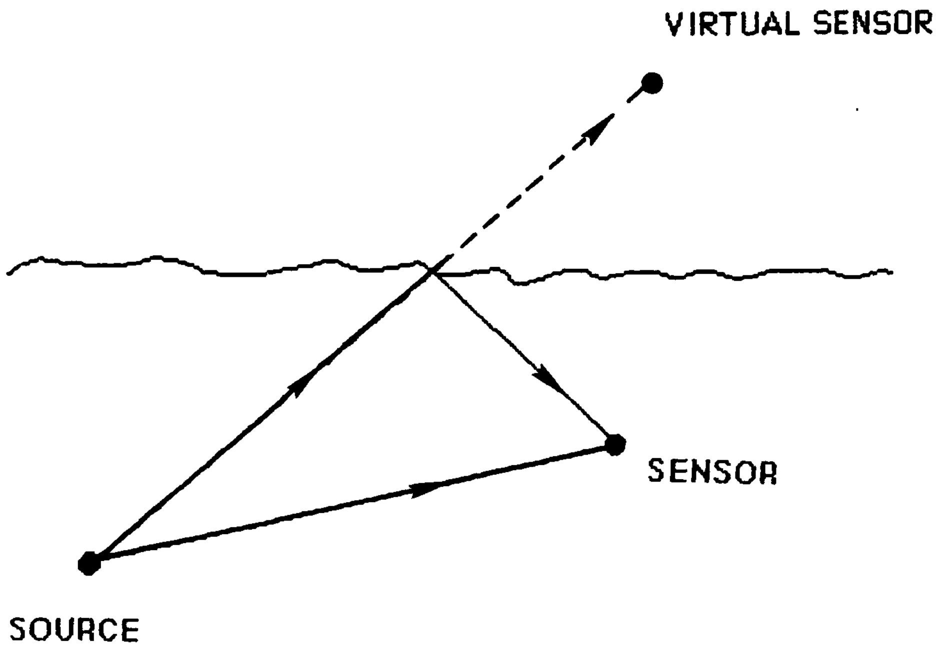


Figure 3.6 The equivalence of a two sensor vertical array in the absence of multipath and a single sensor array in the presence of multipath can be seen in the above diagram.

where $Z_{s2} > Z_{s1}$

then, making the substitution

$$Z_S = \frac{Z_{s2}-Z_{s1}}{2} \tag{3.18}$$

in (3.17) will result in track parameter estimates from inter-sensor delay and doppler estimates. Note that R_{CPA} in this case is measured as the distance of closest approach to the deepest sensor. Also note that Z_T is the depth relative to the midpoint between the sensors.

The various estimates of V, Z_T and R_{CPA} (sensor 1, sensor 2, and inter-sensor measurements) should be combined so as to minimize the variance of the final estimates. The method described in (2.6) is optimal over all linear combinations of the different estimates. In this case, more weight is given to the smaller variance estimates. If the multipath reflection is weak, the inter-sensor measurements are more important. Otherwise, the estimate made from the array with the larger aperture size will have the least variance and therefore more weight. (The aperture size is $2Z_{S_i}$ for the i^{th} sensor, and $Z_{S_i}-Z_{S_j}$ for inter-sensor measurements).

The Horizontal Array

The horizontal array, depicted in Figure 3.4, consists of two sensors at the same depth. As this array is not radially symmetric, intersensor multipath information leads to estimates of bearing angle as well as R_{CPA} and V. These estimates can be combined with estimates from data at individual sensors to form estimates of Z_T, the depth of the target; R_{CPA}, the radius of closest approach; V, the target velocity; X_C and Y_C, the X and Y axis crossings of the target; and θ, the bearing angle. Due to symmetries in the array, however, θ and Y_C can be only determined to within a sign.

The horizontal array has sensors located at

S1: $(-X_s, 0, Z_s)$

S2: $(X_s, 0, Z_s)$.

The target is moving by the sensors at angle θ relative to the X axis, at a velocity V, and a depth of Z_T .

The position of the target is given by,

$$\text{Target:} \quad (Vt\cos\theta + X_T,\ Vt\sin\theta + Y_T,\ Z_T).$$

So the target is at CPA when t=0, define, ctnθ

$$Y_T = X_T \text{ctn}\theta \ .$$

Note that a target travelling along this trajectory will have axis crossings of

$$X_c = X_T(1 + \text{ctn}^2\theta)$$

$$Y_c = Y_T(1 + \tan^2\theta)$$

From (2.5) the inter-sensor delay is given by

$$D_{12}(t) = [(Vt\cos\theta + X_T + X_S)^2 + (Vt\sin\theta + Y_T - Y_S)^2 - (Z_S - Z_T)^2]^{1/2}$$

$$-[(Vt\cos\theta + X_T - X_S)^2 + (Vt\sin\theta + Y_T - Y_S)^2 + (Z_S - Z_T)^2]^{1/2} \quad (3.19)$$

Figure 3.7 shows the cross-correlogram and differential doppler for this case.

By examining the far range behavior of (3.19), an estimate of θ can be obtained. When the target is far from the array, signals emitted from the target arrive at the array from essentially the same direction, θ -- the bearing angle. The intersensor time delay will then be

$$D(|t| \to \infty) = \pm \frac{2X_s}{C} \cos\theta \ ,$$

The bearing angle, θ can therefore be found as:

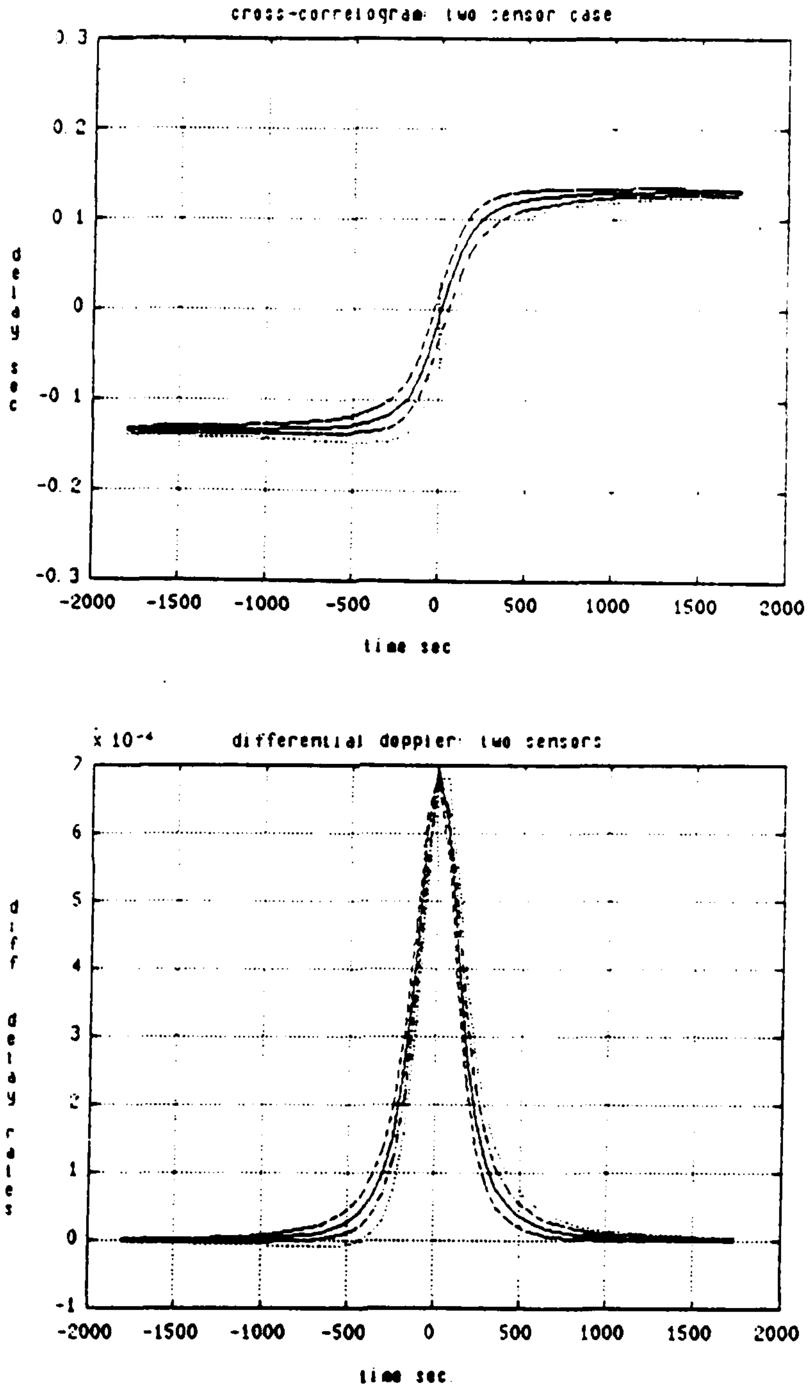


Figure 3.7 The cross-correlogram and differential doppler for a two sensor horizontal array in the presence of a multipath reflection. In this case the target is travelling along a line parallel to the line drawn between the sensors.

$$\cos\theta = \frac{CD(|t|>>|)}{2X_s} \tag{3.20}$$

Examining the intersensor delay at CPA, (at t=0), yields an expression relating X_T and R_{CPA}. Recall,

$$R_{CPA} = (X_T^2 + Y_T^2 + (Z_T - Z_S)^2)^{1/2} \tag{3.21}$$

Using (3.17)

$$CD_{12}(0) = [R_{CPA}^2 + 2X_TX_S]^{1/2} - [R_{CPA}^2 - 2X_TX_S]^{1/2} \tag{3.22}$$

Sovling for R_{CPA},

$$R_{CPA} = [(C^2D_{12}^2(0) - 2X_TX_S)(C^2D_{12}^2(0) - 6X_TX_S)]^{1/4} \tag{3.23}$$

When the relative sensor delay is zero, the target is between the sensors, at $(0, Y_c, Z_T)$. Examining the relative delay rate at this point yields an expression for Y_C. Define

$$t_{Y_C} \triangleq t : D_{12}(t) = 0 \tag{3.24}$$

Recall,

$$\frac{\partial D_{12}}{\partial t} = \frac{V\cos\theta}{C^2D_1}(Vt\cos\theta + X_T + X_S) - \frac{V\cos\theta}{C^2D_2}(Vt\cos\theta + X_T - X_S)$$

$$+ \frac{V\sin\theta}{C^2D_1}(Vt\sin\theta + V_T - X_S) - \frac{V\sin\theta}{C^2D_2}(Vt\cos\theta + X_T - X_S) \tag{3.25}$$

where D_1 and D_2 are the delays from the target to sensor 1 and sensor 2. When $D_{12} = 0$, $D_1 = D_2$, and the target is at $(0, Y_c, Z_T)$. Therefore,

$$\left.\frac{\partial D_{12}}{\partial t}\right|_{t_{Y_c}} = 2X_S \frac{V\cos\theta}{C^2D_1},$$

$$CD_1 = \frac{2X_S \frac{V}{C}\cos\theta}{\dot{D}(t_{Y_c})} \tag{3.26}$$

At t_{Y_c} ,

$$C^2D_1^2 = (Z_T - Z_S)^2 + Y_c^2 + X_S^2$$

and, solving for Y_c

$$Y_c = [X_S^2[(\frac{2V\cos\theta}{C\dot{D}(t_{Y_c})})^2 - 1] - (Z_T-Z_s)^2]^{1/2} \tag{3.27}$$

When the differential delay between sensors is at a maximum, the target is at $(X_c, 0, Z_T)$. Looking at $D_{12}(t_{X_c})$, where

$$t_{X_c} \triangleq \underset{t}{\mathrm{Max}}|D_{12}(t)| , \tag{3.28}$$

yields an expression for X_c . At t_{X_c}, $D_{12}(t)$ is given by

$$CD_{12}(t_{X_2}) = [(X_s+X_c)^2 + (Z_s-Z_T)]^{1/2} - [(X_s-X_c)^2 + (Z_s-Z_T)]^{1/2} \tag{3.29}$$

Solving for X_c ,

$$X_c^2 = \frac{\frac{C^4D_{12}^4(t_{X_c})}{4} + (X_s^2+(Z_s-Z_T)^2)C^2D_{12}^2(t_{X_c}))}{4X_s^2 - C^2D_{12}^2(t_{X_c})}$$

The sign of X_c can be determined by looking at the times of CPA of the individual sensors; X_c will have the sign of the sensor location that had the first CPA. Note, however, if $Z_s \sim Z_T$ or the bearing angle is close to 0, then $D_{12}(t_{X_c}) \sim \frac{2X_s}{C}$ and X_c cannot be determined accurately in this manner.

Velocity can be determined by measuring various times of CPA and axis crossings. Four position-time pairs can be fairly easily measured.

$(X_c,0,Z_T)$ at $t_{X_c} = \underset{t}{\max}\ D_{12}(t)$

$(0, Y_c, Z_T)$ at $t_{y_c} = t{:}D_{12}(t) = 0$

$$(\alpha, \beta, Z_T) \quad \text{at} \quad t_{R_{CPA_1}} = \max_t D_1(t)$$

$$(\alpha + ZX_s \sin\theta\cos\theta, \beta + Z_{X_s}\cos^2\theta, Z_T) \quad \text{at} \quad t_{R_{CPA_2}} = \max_t D_2(t)$$

where $D_1(t)$ and $D_2(t)$ are the differential delays between the direct and multipath signals at sensor 1 and sensor 2; and α, β are constants. Two of the many estimates of velocity are,

$$V - \left(X_c^2 - Y_c^2\right)^{1/2} \tag{3.31a}$$

$$V - \frac{2X_s}{\cos\theta} \frac{1}{t_{R_{CPA_1}} - t_{R_{CPA_2}}} \tag{3.31b}$$

To summarize, estimates of θ, Y_c, X_c, R_{CPA}, and V can be obtained as functions of the inter-sensor delay and doppler (delay rate), Z_T and sensor coordinates. These esitmates, in combination with single sensor multipath data completely describe the track parameters of a target moving along a straight line past the array.

4. PARAMETRIC FIT METHODS OF TRACK PARAMETER ESTIMATION

The parameter extraction techniques presented in the previous chapter relied on knowledge of D(t) and derivatives at specific points in time, namely $|t|<<1$ (near CPA) and $|t|>>1$ (far from CPA). Often times, D(t) is not known accurately in these regions of time (see appendix 1 or section 1.1), and the parameter extraction techniques described in the previous chapter can not be used. This chapter develops parameter extraction techniques based on all available delay information, not just that near and far from CPA. Two different techniques are developed and applied to single sensor, two sensor vertical, and two sensor horizontal arrays.

As the functional form of the delay is known (see section 2), a parametric fit of a model $\hat{D}$ can be made to the measured D. The parameter estimates would then be chosen as,

$$(\hat{X}_T, \hat{Y}_T, \hat{Z}_T, \hat{V}) = \min_{(X_T, Y_T, Z_T, V)} d(D, \hat{D}) \tag{4.1}$$

where, CD is the measured delay, and recall

$$C\hat{D} = [(X_T - Vt)^2 + Y_T^2 + (Z_T - Z_S)^2]^{1/2} - [(X_T - Vt)^2 + Y_T^2 + (Z_T + Z_S)^2]^{1/2}$$

for some distance measure d.

The minimization (4.1) is over a cost function which is non-convex in the track parameters for common distance measures. Therefore, exhaustive search methods must be used in finding a solution. If each of the four parameters were quantized to be one of n values, and there were m data points available, the minimization (4.1) would require O(mn**4) computations; about 25 hours compute time for m=500, n=100, on a 1 mflop computer. To overcome this computational burden, (4.1) can be reformulated so that functions of (X_T, Y_T, Z_T, V) appear as linear coefficeints in a least squares minimization, requiring only O(4m) computations, and less than one second compute time on the same computer. Two linear least squares formulations of (4.1) are developed below. The case of a single sensor listening to a target in the presence of a multipath reflection is considered first, and later generalized

to the case of two sensors.

4.1 EQUATION ERROR APPROACH

This method is an equation error-like approach. Recall,

$$D = P-Q, \tag{4.2}$$

where,

$$P = [(X_T-Vt)^2 + Y_T2 + (Z_T+Z_s)^2]^{1/2}/C$$

$$Q = [(X_T-Vt)^2 + Y_T2 + (Z_T-Z_s)^2]^{1/2}/C$$

What is desired is a relation in terms of D, P^2, and Q^2 so that functions of track parameters appear as linear coefficients which can be fit using least squares techniques. Manipulating (4.2) yeilds the following relationships,

$$D^2 = P^2 + Q^2 - 2PQ \tag{4.3}$$

$$(D^2 - P^2 - Q^2)^2 = 4P^2Q^2 \tag{4.4}$$

$$(CD)^4 - 4V^2(cDt)^2 + 8VX_T(C^2D^2t) - 4(Y_T2+Z_T2+Z_s^2+X_T2)(CD)^2 + 16Z_T^2\, Z_s^2 = 0 \tag{4.5}$$

With values of D(t) specified at m time instances, solving for the coefficeints of (4.5) which minimize,

$$P^* = \min_A \|\hat{f}-f\|^2 \ , \ \hat{f} = \phi A \tag{4.6}$$

$$\phi = \begin{bmatrix} (CD(t_1)t_1)^2 & C^2D(t_1)^2t_1 & C^2D(t_1)^2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ (CD(t_m)t_m)^2 & C^2D(t_m)^2t_m & C^2D(t_m)^2 & 1 \end{bmatrix}$$

is a standard linear least squares problem.

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad f = \begin{bmatrix} (CD(t_1))^4 \\ \vdots \\ (CD(t_m))^4 \end{bmatrix}$$

The optimal set of coefficeints, A* is given by,

$$A^* = (\Phi^T \Phi)^{-1} \Phi^T f \tag{4.7}$$

and the resulting track parameter estimates are,

$$\begin{bmatrix} V \\ X_T \\ Y_T \\ Z_T \end{bmatrix} = \begin{bmatrix} \frac{1}{4}\sqrt{a_1} \\ -\frac{1}{Y} a_2/\sqrt{a_1} \\ \frac{1}{4}\left(a_3 - \sqrt{a_4}/4Z_s - Z_s^2 + \frac{a_2^2}{a_1} \cdot \frac{1}{16}\right)^{1/2} \\ \frac{\sqrt{a_4}}{4Z_s} \end{bmatrix} \tag{4.8}$$

SENSITIVITY ISSUES

Upon closer examination of (4.1) or (4.6), one finds that there are many sets of parameters (X_T, Y_T, Z_T, V) that result in roughly the same D(t). Indeed, the differences in D(t) between the case of a target moving slowly close by a sensor and the case of the target moving more quickly further from the sensor are subtle, and are small compared to the variance in estimating D(t).

This ambiguity shows up in the eigen-structure of the matrix used in finding the track parameter estimates via (4.6). For typical values of the track parameters, the matrix,

$$\Phi^T \Phi$$

has an eigenvalue which is o(10**-4) smaller than the rest. The coefficeint array, A, will therefore be very sensitive to any noise in D(t).

The eigenvector corresponding to the troublesome eigenvalue is roughly,

$$ev = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Therefore, specification of V and an appropiate modification of (4.6) will result in track parameter estimates that are relatively insensitive to noise in the measurements of D(t). With V specified, the modifications to (4.6) are,

$$A^* = \min_A \|\hat{f} - f\|^2, \quad \hat{f} = \Phi A \tag{4.9}$$

$$\Phi = \begin{bmatrix} C^2 D(t_1)^2 t_1 & C^2 D(t_1)^2 & 1 \\ \vdots & \vdots & \vdots \\ C^2 D(t_m)^2 t_m & C^2 D(t_m)^2 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad f = \begin{bmatrix} (CD(t_1))^4 - 4V^2 (CD(t_1) t_1)^2 \\ \vdots \\ (CD(t_m))^4 - 4V^2 (CD(t_m) t_m)^2 \end{bmatrix}$$

The least squares solution for A is then given by,

$$A^* = (\Phi^T \Phi)^{-1} \Phi^T f \tag{4.10}$$

and the track parameter estimates are,

$$\begin{bmatrix} X_T \\ Y_T \\ Z_T \end{bmatrix} = \begin{bmatrix} -a_1/8V \\ (a_2/4 + (a_1/8V)^2 - Z_s^2 - a_3/16Z_s^2)^{1/2} \\ \sqrt{a_3}\ /4Z_s \end{bmatrix} \tag{4.11}$$

4.2 TRANSFORM APPROACH

In this approach, a transformation, which is a function of a depth estimate, is applied to D(t). When the depth estimate is accurate, D(t) is transformed into a second order polynomial whose coeffeceints are easily fitted.

The transformation is given by,

$$T = \left(CD \pm \hat{Z}_T \frac{4Z_s}{CD}\right)^2 \tag{4.12}$$

using (2.1) for D(t),

$$T = \frac{(1\pm\alpha)^2}{4}[(X_T-Vt)^2+Y_T^2+(Z_T\pm Z_s)^2] + \frac{(1\mp\alpha^2)^2}{4}[(X_T-Vt)^2+Y_T^2+(Z_T\mp Z_s)^2]$$

$$+ \frac{(1-\alpha)^2}{2}[((X_T-V_T)^2+Y_T^2+(Z_T+Z_s)^2)((Y_T-Vt)^2+Y_T^2+(Z_T-Z_S)^2)]^{1/2} \tag{4.13}$$

where

$$\alpha = \hat{Z}_T/Z_T$$

When $\alpha \sim 1$, i.e. $\hat{Z}_T \sim Z_T$, (4.13) reduces to,

$$T = (X_T-Vt)^2 + Y_T^2 + (Z_T \pm Z_s)^2 + O(\alpha-1) \tag{4.14}$$

Fitting a polynomial to (4.14), by least squares, for instance, gives,

$$\hat{V} = \sqrt{a_1}$$

$$\hat{X}_T = -\frac{1}{2\sqrt{a_1}} a_2 \tag{4.15}$$

$$\hat{Y}_T = (a_3 - (\hat{Z}_T \pm Z_s)^2 - \hat{X}_T^2)^{1/2}$$

where a_1, a_2, a_3 are chosen to minimize

$$\| a_1 t^2 + a_2 t + a_3 - T_{\hat{Z}_T}(D(t)) \|^2 ,$$

and $\hat{Z}_T$ is picked a priori.

The optimal value of $\hat{Z}_T$, and therefore $(\hat{X}_T, \hat{Y}_T, \hat{V})$, can be found by preforming a line search over $\hat{Z}_T$, using the following as possible error criteria:

$$\text{error} = \sum_i [(a_1 t_i^2 + a_2 t_i + a_3) - T_{\hat{Z}_T}(D(t_i))]^2 \tag{4.16}$$

$$\text{or, error} = \sum_i (\hat{D}(t) - D(t))^2,$$

where $\hat{D}(t)$ is given by (2.1) using, $(\hat{X}_T, \hat{Y}_T, \hat{Z}_T, \hat{V})$.

SENSITIVITY ISSUES

As expected, the error, (4.16) achieves a minimum only for those cases where D(t) is known very precisely (one part in ten thousand for typical values of the track parameters). Since V can be reliably determined from other measurements, a line search over $\hat{Z}_T$ can be peformed using,

$$\text{error} = (\hat{V}(\hat{Z}_T) - V)^2 \tag{4.17}$$

instead of (4.16) as an error criterion. As the transformation T results in a one to one relationship between $\hat{Z}_T$ and V, there are no uniqueness problems in the determination of $\hat{Z}_T$. Knowledge of V, therefore, allows use of (4.12) and (4.17) in estimating the remaining track parameters from noisy measurements of D(t).

4.3 TWO SENSOR EXTENSIONS

In this section, methods for adapting the above techniques to inter-sensor measurements in two sensor arrays are presented. In addition, methods for combining information from two sensors to resolve ambiguities in determining the track parameters are developed.

MODIFICATION FOR INTER-SENSOR DELAY

It is fairly straightforward to modify (4.6) or (4.12) so that they can be used with inter-sensor delay information (instead of multipath delay information). If the sensors are placed in a vertical array, no modification of either method is needed, only estimated target depth needs to be adjusted (see 3.18). With the sensors placed in a horizontal array, changes to the two methods must be made.

The inter-sensor delay in the case of a horizontal array is still written as,

$$D = P-Q \tag{4.18}$$

where, in this case,

$$P = [(X_T-Vt\cos\theta - X_s)^2 + (Y_T-Vt\sin\theta)^2 + (Z_T-Z_s)^2]^{1/2}$$

$$Q = [(X_T-Vt\cos\theta + X_s)^2 + (Y_T-Vt\sin\theta) + (Z_T-Z_s)]^{1/2}$$

and

$$(CD)^4 + (cDt)^2(-4V^2) + 8(cD)^2t \cdot (VX_T\cos\theta + VY_T\sin\theta)$$

$$+ (CD)^2 \cdot (-Y)(X_T^2+ X_s^2 + Y_T^2 + (Z_s-Z_T)^2) + t^2\cdot(16X_s^2V^2\cos^2\theta)$$

$$+ t \cdot (-16X_s^2X_TV\cos\theta) + (16X_s^2X_T^2) = 0$$

With the exception that there are six variables to be determined instead of four, solving for the coefficeints of (4.18) is equivalent to solving for the coefficeints of (4.2). The track parameters in (4.18) can be determined by the method described in (4.6). Additionally, any a priori knowledge of track parameters can be incorporated in a manner similiar to (4.9).

The transformation,

$$T = \left(CD \pm \frac{\gamma}{CD}\right)^2 \tag{4.19}$$

will again yeild a polynomial for the correct choice of γ. The coefficeints of the polynomial can then be fitted, determining the remaining track parameters. Unfortunately, the optimal choice of γ is,

$$\gamma^* = 2X_S(X_T - Vt\cos\theta) \tag{4.20}$$

and the search for γ is now over a two dimensional surface, which, without a good error criterion, will not yield accurate or computationally efficient results.

COMBINING ESTIMATES FROM TWO SENSORS

Even though there are many sets of track parameters resulting in essentially the same multipath delay curve at a sensor, these same sets of track parameters will not necessarily generate similiar delay curves at another sensor location. Therefore, if information from two sensors is available, ambigiuties in track parameter estimates can be resolved by looking for track parameter estimates which are consistent at the two sensor locations.

If data is available from two sensors placed in a vertical array, sets of estimates, parameterized by V (or Z_T), can be constructed using either (4.9) or (4.12). A line search can then be performed over V (or Z_T) for the set of track parameters minimizing the error,

$$\text{error} = [\hat{Y}_T(i,1) - \hat{Y}_T(i,2)]^2 \tag{4.21}$$

where $\hat{Y}_T(\ell,m)$ is the Y_T estimate from sensor m based on $\hat{V} = V_\ell$ or, equivalently,

$$\text{error} = [\hat{r}_{CPAO}(i,1) - \hat{r}_{CPAO}(i,2)]^2 \tag{4.22}$$

where $\hat{r}_{CPAO}(\ell,m)$ is the m^{th} sensor CPA range estimate to the point (0,0,0) based on $\hat{V} = V_\ell$.

If the data is from two sensors placed in a horizontal array, sets of estimates, parameterized by V (or Z_T), based on individual sensor data can be generated by (4.9) or (4.12). Estimates of the bearing angle, parameterized by v (or Z_T) can be made based on inter-sensor data using (4.18). Track parameter estimates can then be chosen by finding the set of estimates producing the most consistent $(\hat{r}_{CPA1}, \hat{r}_{CPA1}, \hat{\theta})$, i.e., the set $(\hat{r}_{CPA1}, \hat{r}_{CPA1}, \hat{\theta})$ minimizing the error,

$$\text{error} = \left[\sin\theta - \frac{\sqrt{r^2_{CPA_1} - Z^2_s} - \sqrt{r_{CPA_2} - Z^2_s}}{2X_s}\right]^2 \tag{4.23}$$

4.4 COMPUTER SIMULATION RESULTS

In this section, results of the parameteric fit methods of track parameter estimation applied to simulated data are discussed. The equation error method and the transform method are applied to delays simulated for one and two sensor arrays.

DATA

The data, $\hat{D}$ was generated by calculating the true D(t), using (2.1), and adding white gaussian noise. Since D(t) estimates tend to be less certain near CPA (see appendix 1), the additive noise is scaled in proportion to the delay value in the case of single sensor multipath delay curves, and in the case of 2-sensor horizontal arrays, the noise is of constant variance.

Figure 4.1 shows examples of the simulated data. In Figure 4.1, the solid line is the true value of the delay; the dots are the data points used. The signal to noise ratio in these cases is about 20.

EQUATION ERROR METHOD

The equation error method was applied to data generated using,

Target: (5t, 1000, 200)

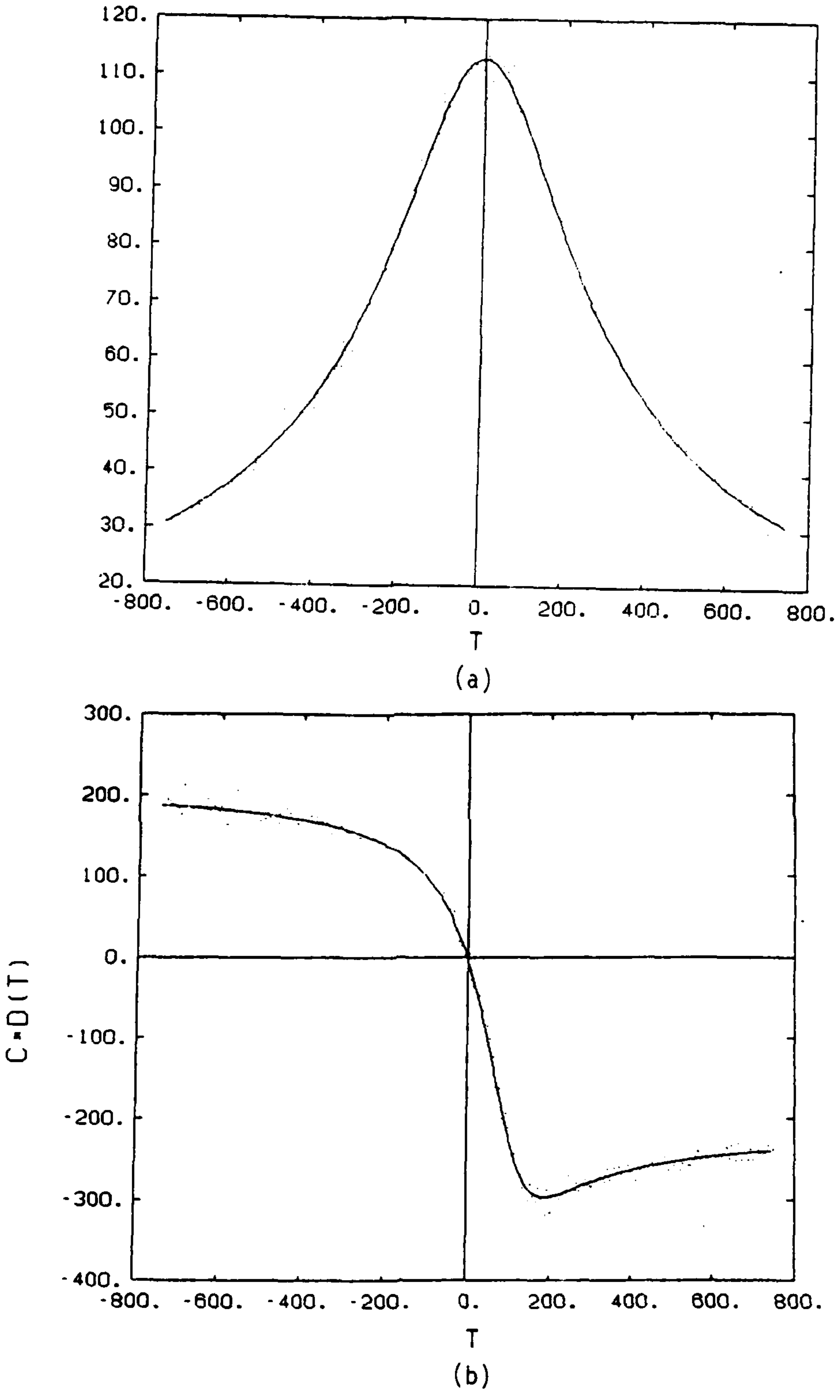


Figure 4.1 (4.1a) The simulated (dotted) and real (solid) differential delay curves for a 2-sensor horizontal array.

(4.1b) The same curves plotted for a single sensor with a multipath reflection.

Sensor: (0, 0, 300)

When no additive noise is present, the track parameters are estimated to one part in 10^4. With a SNR of about 20, however, the estimates are off by factors of 100, due to the sensitivity problem discussed earlier.

If the value of V is assumed known, the equation error method can be applied to the noisy data with much improved results. Figure 4.2 shows track parameter estimates as functions of V for the equation error method applied to a data set with a SNR of 20. Note that at the correct value of V, the track parameter estimates are close to the true values.

Sensitivity

To illustrate the sensitivity problems, two delay curves, generated using different sets of track parameters have been plotted. Figure 4.3a and 4.3b show delay curves plotted using the estimated track parameters generated with V=5 and V=10. Also plotted are data sets with SNR's of 20. Both curves fit the data well. In fact, the differences between the two curves themselves are negligible. Therefore, in this case additional knowledge (knowledge of V, for instance) is required to make a unique choice of track parameter estimates.

TRANSFORM METHOD

The transform method was used to generate the track parameter estimates plotted as functions of $\hat{Z}_T$ in Figure 4.4. Figure 4.4a shows estimates made from noiseless data. Estimates made from data with SNR=20 are essentially the same. The error, which is the norm of the difference between the transformed delay curve and the closest quadratic, is plotted as a function of $\hat{Z}_T$ for the two cases in Figure 4.4b.

As functions of $\hat{Z}_T$, the track parameter estimates for the two cases are very similiar, and assume values close to the true track parameter values for the correct choice of $\hat{Z}_T$. The error curves, however, indicate that practically any noise in estimating D(t) will lead to ambiguities in determining track parameter estimates. As V is usually available through

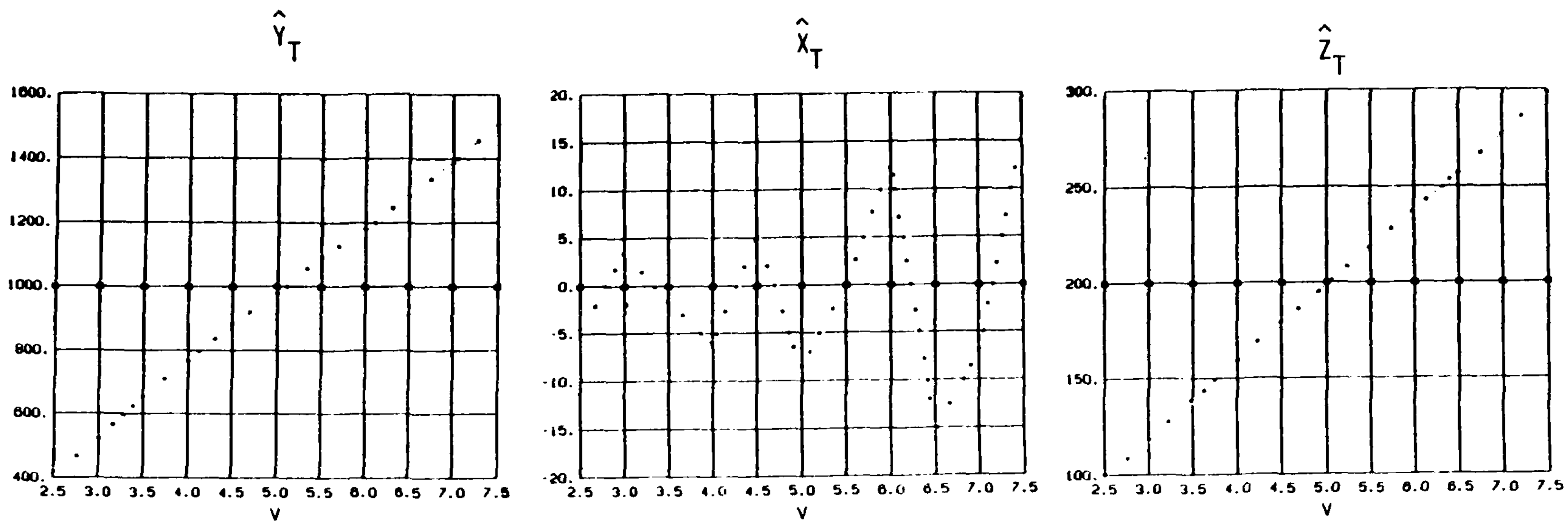


Figure 4.2 Track-parameter estimates as functions of an a priori estimate of velocity. The true parameter values are marked by squares, and the true velocity is 5 m/s.

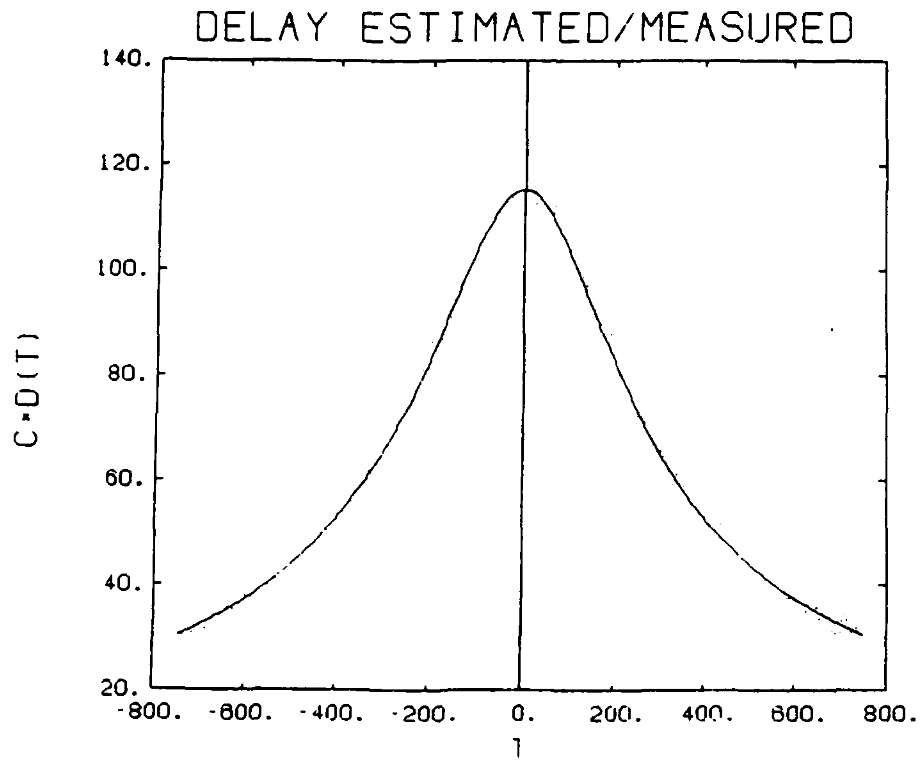


Figure 4.3a Estimated delay curve (solid) using V=5 m/s and the resulting track parameters estiated obtained with the equation error method. The dots are the original data set.

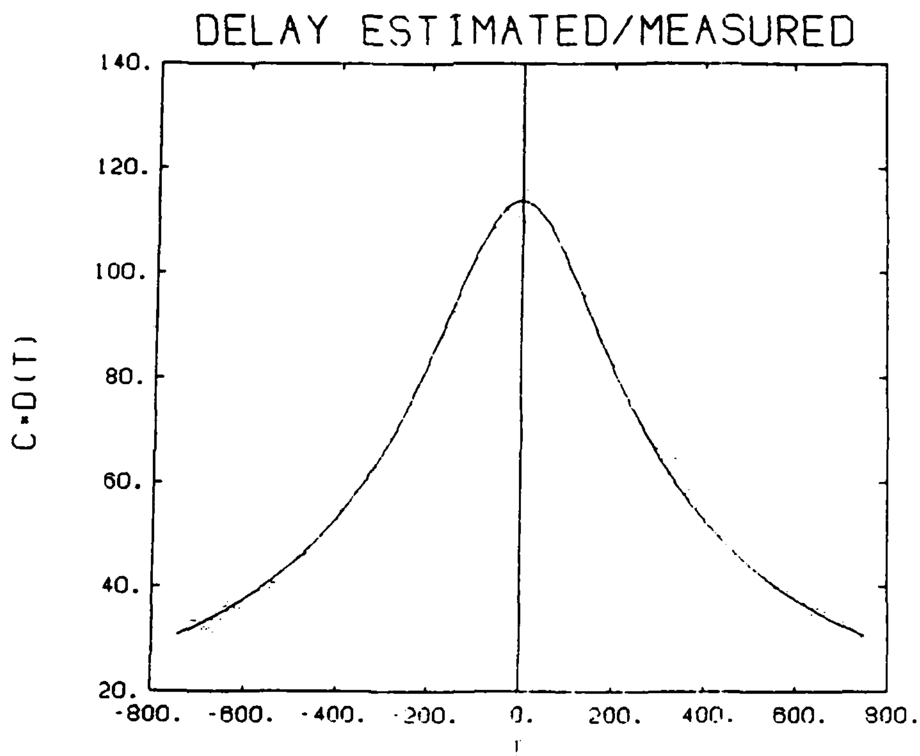


Figure 4.3b The same as 4.3a using V=10 m/s in the equation error method. The actual velocity for this example was V=5 m/s. Note the similarity between the two estimates.

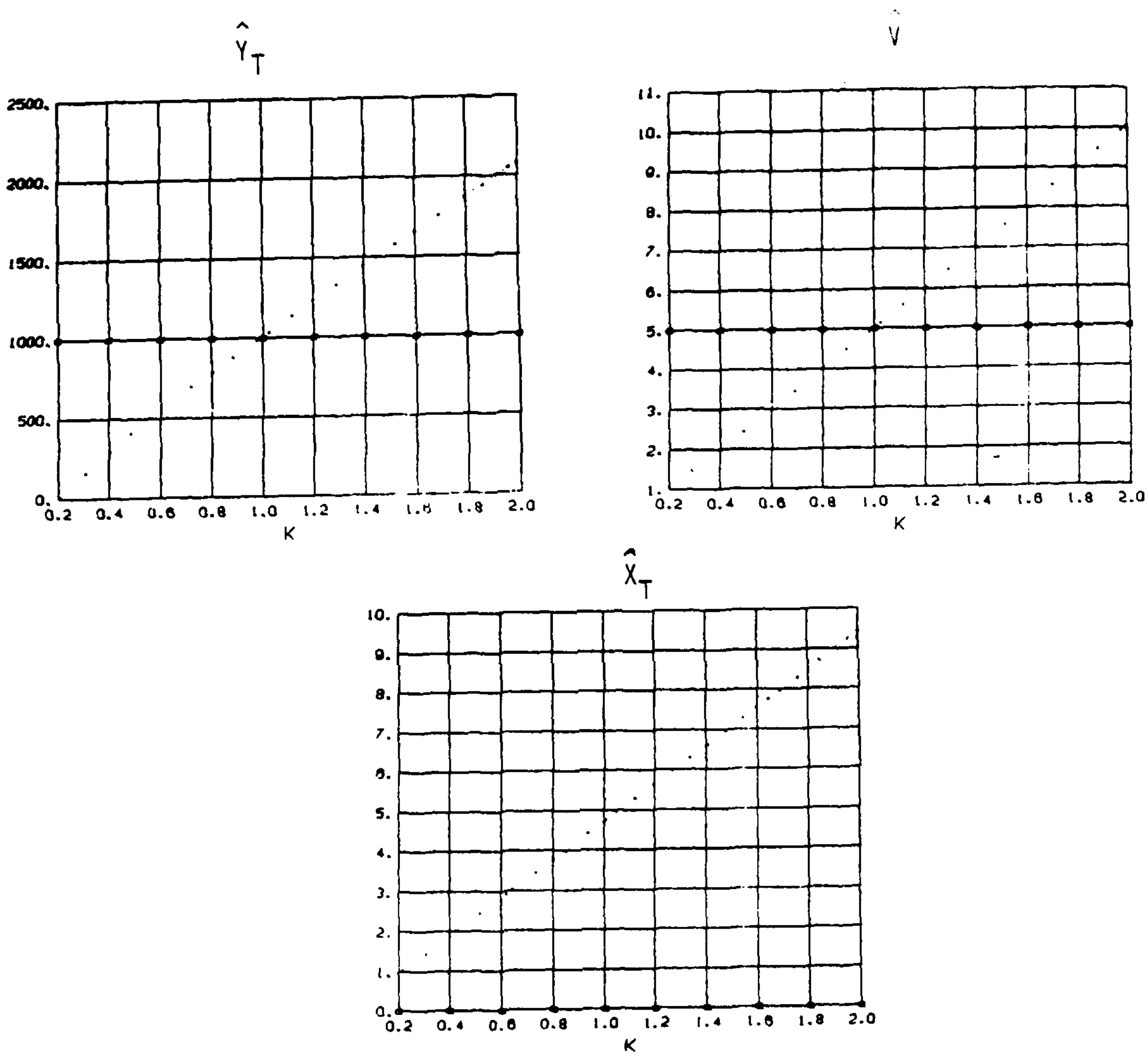


Figure 4.4a Track parameter estimators as functions of $K = \dfrac{\hat{Z}_T}{Z_T}$ using the transform method. The true track parameter values are marked by squares.

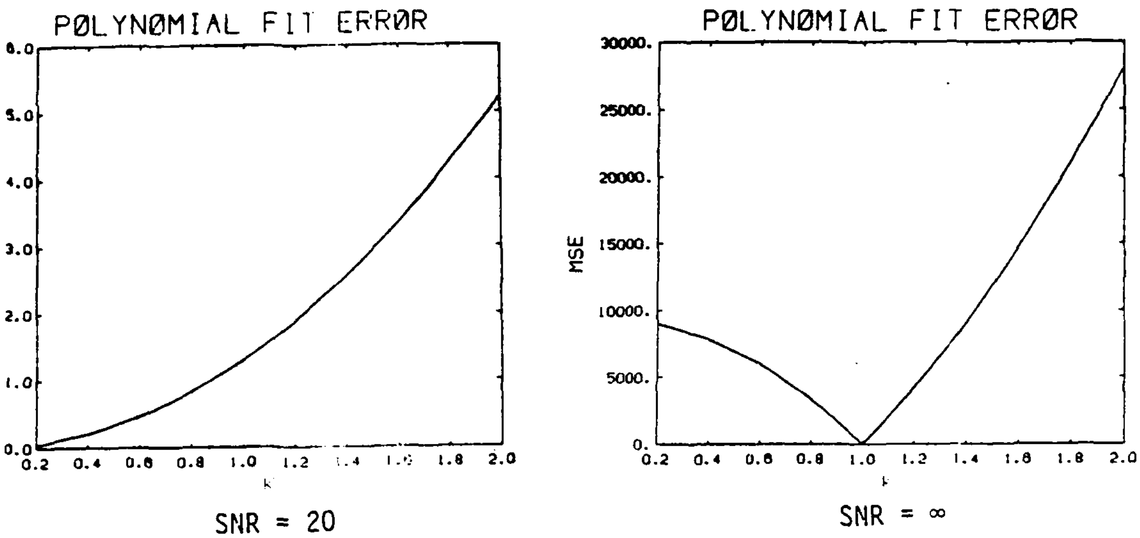


Figure 4.4b The polynomial fit error for SNR's of ∞ , and 20 are plotted here.

other means, the track parameters can be determined by choosing $\hat{Z}_T$ as the one which gives the correct value of V and using $\hat{Z}_T$ to determine the remaining track parameters.

TWO SENSOR EXTENSIONS

When data from two sensors are available, ambiguities in choosing track parameter estimates can sometimes be resolved by looking for consistencies between the two sets of estimates. In a vertical array, delay information alone can determine track parameter estimates when,

$$Y_T \leq O(10 \cdot |Z_{S1}-Z_{S2}|) \text{ and } \frac{\text{VarD}}{\text{ED}} \leq .1$$

and in a horizontal array when,

$$Y_T \leq O(10 \cdot |X_{S1}-X_{S2}|) \text{ and } \frac{\text{VarD}}{\text{ED}} \leq .1$$

Both methods were applied to two sensor data satisfying the above constraints. Computer simulation results are given below.

Vertical array

Both methods were used to estimate track parameters from a two sensor vertical array. For the simulations, the sensors were placed at,

Sensor 1: (0,0,300)

Sensor 2: (0,0,100)

with a target at,

Target: (5t, 1000, 200)

and SNR=20.

The equation error method was used to generate track parameter estimates

as functions of V at each sensor. The error in $\hat{Y}_T$ (4.21) is plotted in Figure 4.5 for two different simulations. For this SNR and $\hat{Y}_T$, the minimum of the error function will occur reasonably near the correct value of V.

Figure 4.6 shows track parameter estimates as functions of Z_T at each sensor. These estimates were generated using the transform method. Figure 4.7 shows CPA to (0,0,0) error, (4.22) as a function of Z_T. Again, for these values of SNR and $Y_T \lesssim 0(10 \cdot |Z_{S1} - Z_{S2}|)$, the minimum of the error curve will occur near the true value of Z_T.

Note that in the above simultaions the inter-sensor delay measurements were not used. Using this information further, estimates of $\hat{Y}_T$ or CPA to (0,0,0) as functions of V or Z_T could be used to lower the variance in determining V or Z_T.

Horizontal Array

Track parameters were estimated based on data from a horizontal array with sensors at

Sensor Locations: (±150,0,300)

In this case, the track parameters were given by

Target: $V = 5$, $\theta = \pi/4$, $X_T = 0$, $Y_T = 650$, $Z_T = 200$

The equation error method was used to estimate track parameters at each sensor as functions of V. The bearing angle was estimated as a function of V from inter-sensor data also using the equation error method. The error criterion, (4.23) was used to determine the best estimate of V (see Figure 4.8). This error will reliably give good estimates of V, and therefore the remaining track parameters, since the inter-sensor angle estimate is an increasing function of V, whereas the individual sensor estimate of angle is a fairly constant function of V.

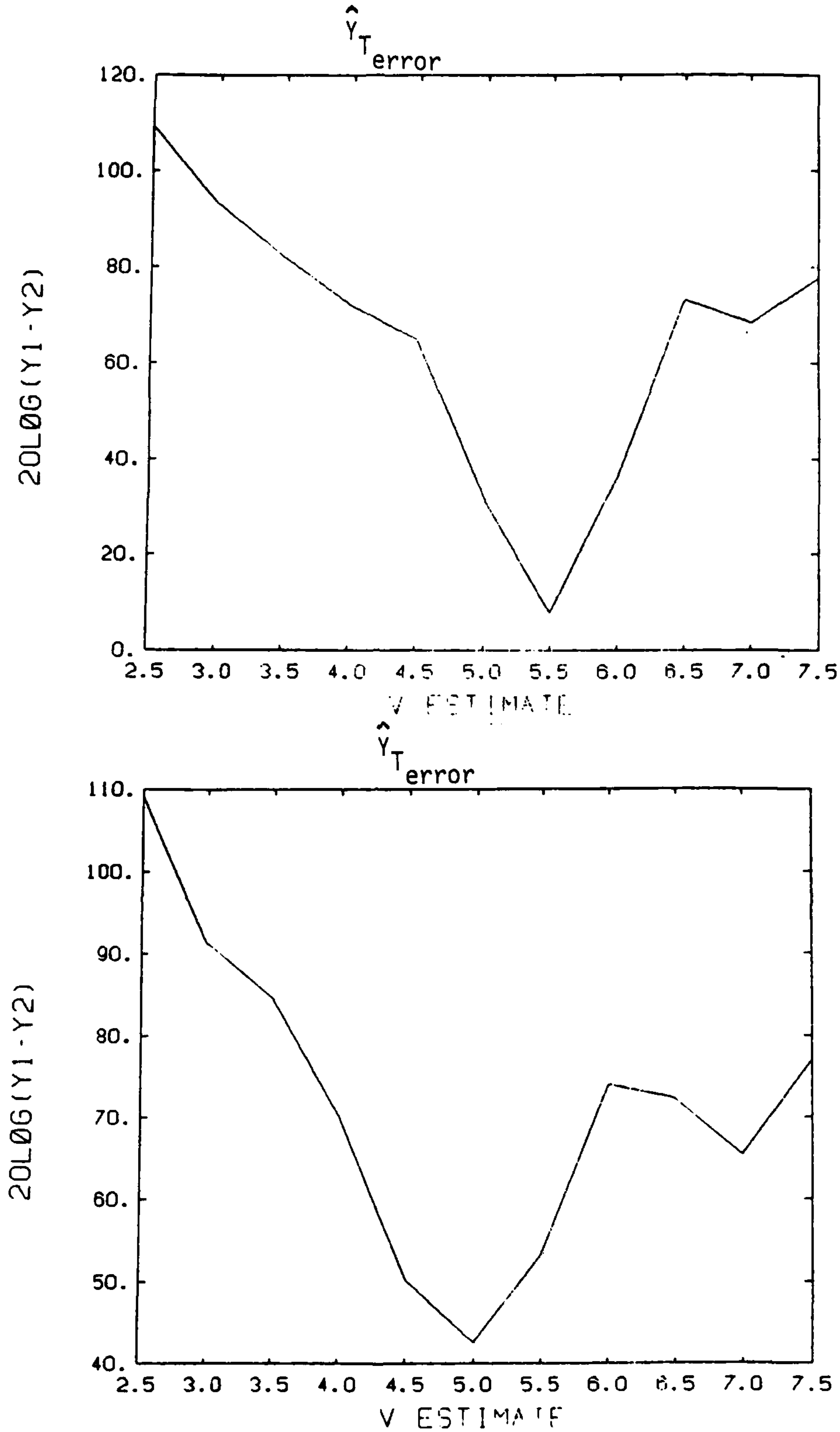


Figure 4.5 Error in $\hat{Y}_T$, (4.21), as a function of a velocity estimate for two cases. The true value of V is V=5 m/s.

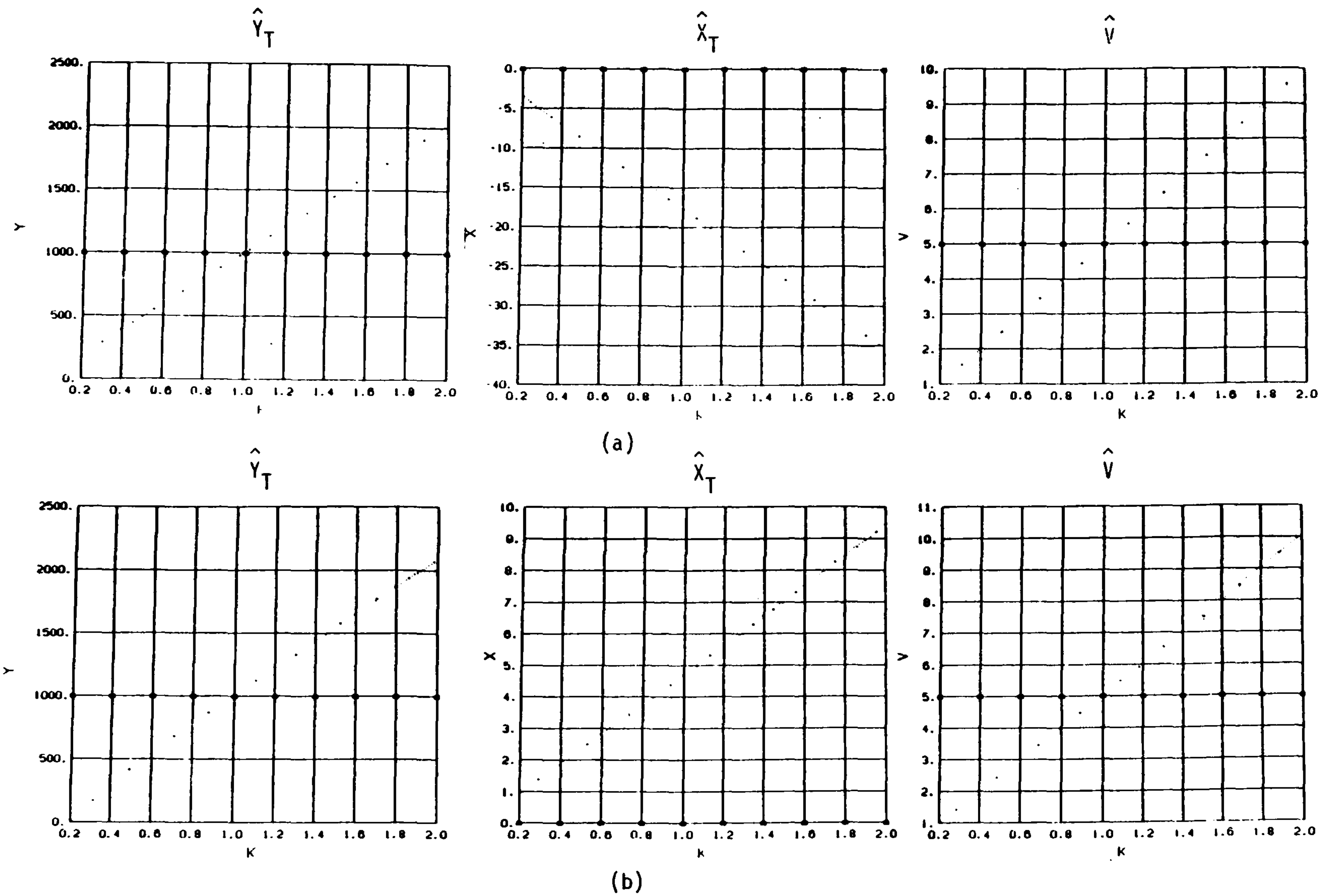


Figure 4.6a Track parameter estimates as functions of $k = \frac{\hat{Z}_T}{Z_T}$ for sensor 1, $Z_{S_1} = 300$ m

Figure 4.6b The same estimates made from sensor 2, $Z_{S_2} = 100$ m .

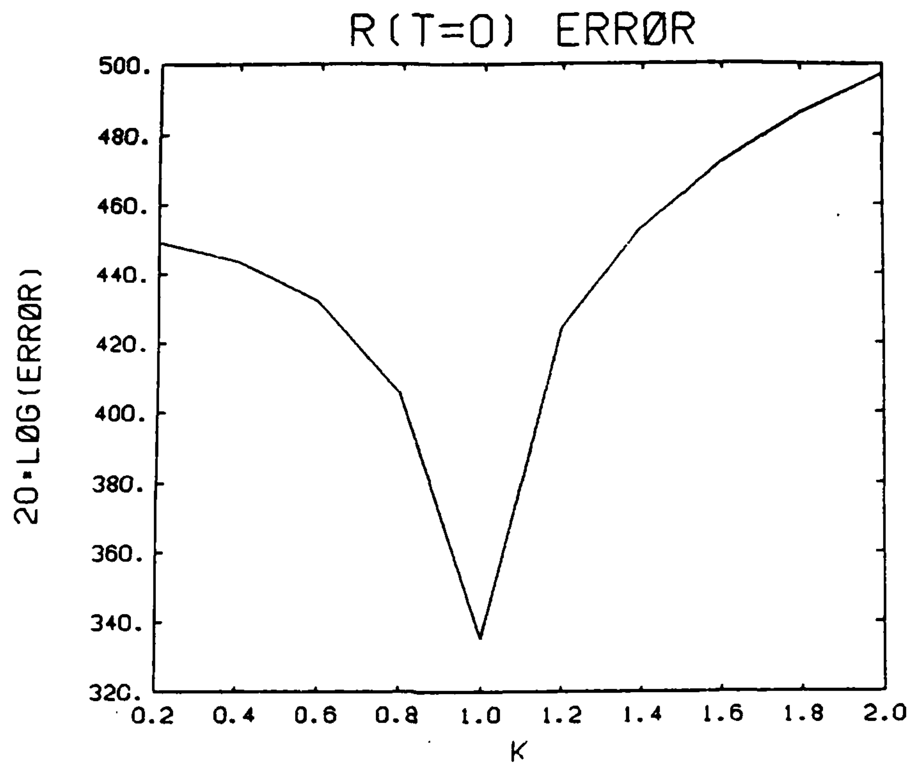


Figure 4.7 CPA error for the estimates shown in Figure 7

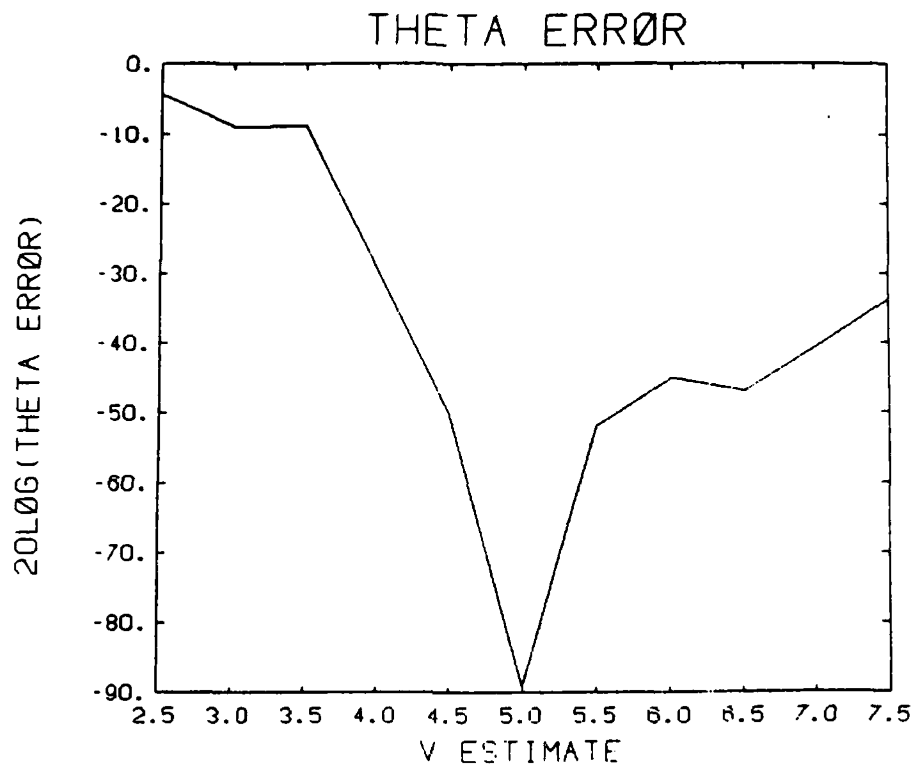


Figure 4.8 The error in θ estimates as a function of velocity estimates based on intersensor and individual sensor estimates. The true velocity is 5 m/s.

REMARKS

In the case of a single sensor in the presence of a multipath reflection, both the equation error method and the transform method yeild accurate results if the velocity or depth are known independently. In the case of a two sensor array, depth or velocity need not be known a priori, but can be chosen using consistency criteria.

5. CONCLUSION

In this report, techniques were developed for extracting track parameter information using sono-bouy data. In the case of a single sensor and a target moving in the presence of a multipath reflection, it was shown that differential delay and doppler information could be obtained and used to find reasonably accurate estimates of the depth, range, and velocity of the target. In the case of two sensors, additional range and velocity estimates can be obtained from intersensor differential delay information. These estimates can be combined to form lower variance estimates of these parameters. Additionally, depending on the sensor locations, further estimates of depth and/or estimates of bearing angle may be found.

As a final note, it is worth mentioning that a significant effort was spent developing software to process sonobouy data. The software developed included correlogram-based TDOA estimators -- SCOT, PHAT, and ML [4], and the ADEC linetracking algorithm. Listings are included in appendix 4. A real data example is presented in appendix 2.

REFERENCES

1. "Principles of Underwater Sound for Engineers," pp. 187-233, Robert J. Urick, McGraw-Hill 1967.

2. "Multipath Delay Estimation," Benjamin Friedlander and J. O. Smith, SCT Technical Report 5466-04, December 1983.

3. "On the Cramer-Rao Bound for Time Delay and Doppler Estimation," Benjamin Friedlander, IEEE Transactions on Information Theory, Vol. IT-30, No. 3, May 1984.

4. "The Generalized Correlation Method for Estimation of Time Delay," Charles H. Knapp and G. Clifford Carter, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-24, No. 4, August 1976.

APPENDIX 1

DIFFERENTIAL DOPPLER EFFECTS ON CORRELATION

Appendix 1: DIFFERENTIAL DOPPLER EFFECTS ON CORRELATION

If two versions of the same signal have relative doppler shifts, the peak of the resulting cross correlation will be reduced compared to that of the cross correlation of the signals without a relative doppler shift. This decorrelation causes fading of correlogram lines near CPA, where differential doppler is greatest.

To examine this effect, assume X(t) is a zero mean, gaussian bandpass process, with bandwidth B, and center frequency F_0. The autocorrelation of X is given by

$$R_X(\tau) = R_0 \frac{\sin \pi B\tau}{\pi B\tau} \cos 2\pi\tau \qquad \text{(A1.1)}$$

The mean correlation output, μ, of the signal X(t) with itself is given by

$$\mu = E\{\frac{2}{T} \int_0^{\frac{T}{2}} X(t)X(t)dt\}$$

$$\mu = R_0 \qquad \text{(A1.2)}$$

The mean correlator output, μ of the signal X(t) with its doppler shifted version is given by

$$\mu = E\{\frac{2}{T} \int_0^{\frac{T}{2}} X(t)X(\alpha t)dt\} \qquad \text{(A1.3)}$$

which is not in general as large as R_0.

Interchanging expectation and integration, and substituting (A1.1) for $E(X(t)X(t+\tau))$,

$$\mu = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} R_0 \frac{\sin \pi B|1-\alpha|t}{\pi B|1-\alpha|t} \cos 2\pi f_0|1-\alpha|t \; dt \qquad \text{(A1.4)}$$

$$\frac{\mu}{R_0} = \frac{1}{\pi BT|1-\alpha|} \int_0^{\frac{\pi BT|1-\alpha|}{2}} \frac{\sin X}{X} \cos \frac{2f_0}{B} X \, dX$$

$$\frac{\mu}{R_0} = \frac{1}{\pi BT|1-\alpha|} \left[\text{sinc}\left(\frac{\pi BT|1-\alpha|}{2}\left(\frac{2f_0}{B} + 1\right)\right) - \text{sinc}\left(\frac{\pi BT|1-\alpha|}{2}\left(\frac{2f_0}{B} - 1\right)\right)\right] \quad \text{(A1.5)}$$

Plots of mean correlator output as a function of doppler shift appear in Fig. (A1.1). Figure (A1.2) is a plot of a typical doppler history for a target moving in a straight line by an array. Several features are easily seen in Figure (A1.1). First, as the relative doppler shift increases, the correlator output decreases. Second, as integration time increases, the correlator output decreases. Finally, as the bandwidth of the signal increases relative to the center frequency, the correlator output increases.

The performance of the correlation method of time delay estimation is greatly effected by nonzero differential doppler. Fortunately, due to the relatively slow speeds of submarines, differential doppler is only likely to cause problems except near the time of CPA. See, for instance, Figure A2.1.

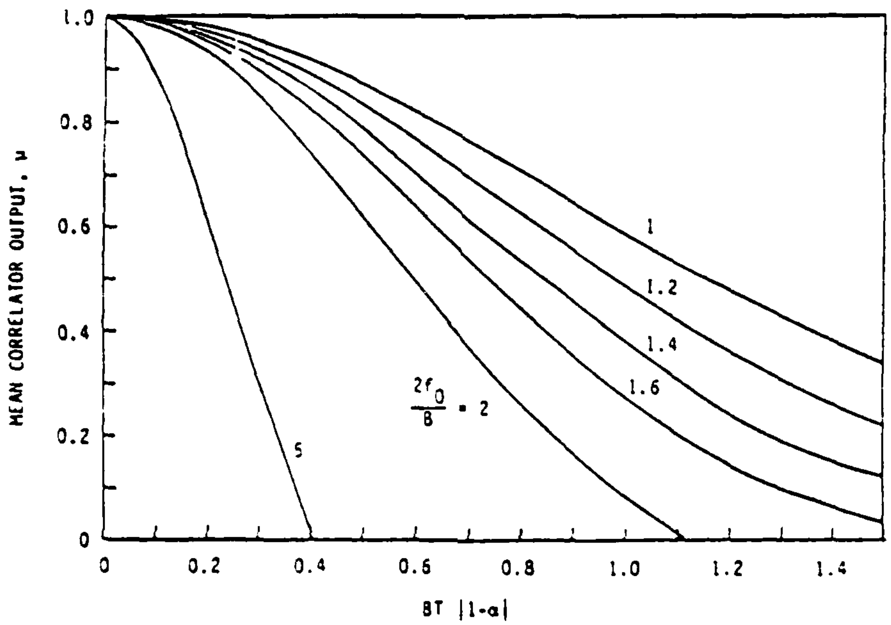


Figure A1.1 Mean correlator output as a function of doppler shift, α. f_0 is the center frequency, and β is the bandwidth of the input process. T is the integration time.

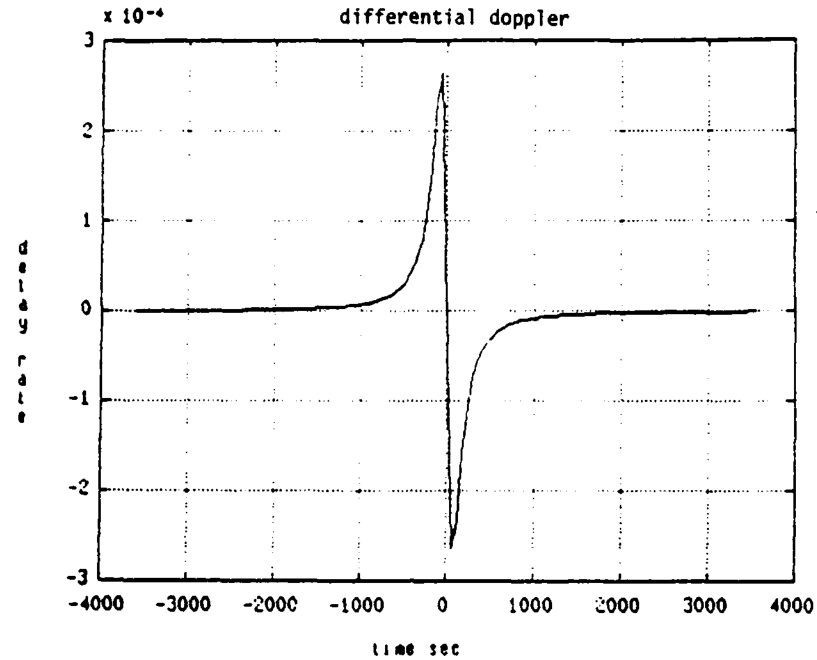


Figure A1.2 Example of the doppler history of a target passing by a sensor. CPA occurs at t=0.

APPENDIX 2

REAL DATA EXAMPLE

Appendix 2: **REAL DATA EXAMPLE**

This appendix presents an example of track parameter estimation from a real data set taken from a horizontal array. The track parameter estimation techniques described in Chapter 3 will be used. In this case, θ, V, $\dot{Z}_t$ and R_{CPA} will be estimated from the data and knowledge of the sensor locations.

The data was taken from an array with sensors located at (± 136m, 0, 308m). Figure A2.1 shows R_{11}, R_{12}, and R_{22} , the correlograms formed using the correlation method of time delay estimation. R_{11} and R_{22} have been energy normalized so that $r_i(0) = 1 \ \forall_i$; R_{12} has been SCOT normalized [4]. As the data set was of exceptional quality, the differential delay information needed for track parameter estimation can be measured without further processing. The immediately measurable parameters are listed below:

$$D_{12}(|t| \to \infty) \sim .187 \text{ sec} \tag{A2.1}$$

$$\left.\frac{\partial D_{12}(t)}{\partial t}\right|_{t:D_{12}(t)=0} \sim \left(\frac{.5 \text{ sec}}{200\text{sec}}\right) = .0025$$

$$t_{R_{CPA_1}} - t_{R_{CPA_2}} \sim 35 \text{ sec}$$

$$D_1(0) \sim .076 \text{ sec}$$

$$D_2(0) \sim .074 \text{ sec}$$

$$\hat{\rho}_1 \sim 4.9$$

$$\hat{\rho}_2 \sim 5.1$$

The relevant equations from the text are as follows:

$$\cos\theta = \frac{CD(|t|>>1)}{2X_s} \tag{A2.2}$$

$$\frac{\dot{Z}_T}{V} = \hat{\rho}\,\frac{C}{2Z_s} \tag{A2.3}$$

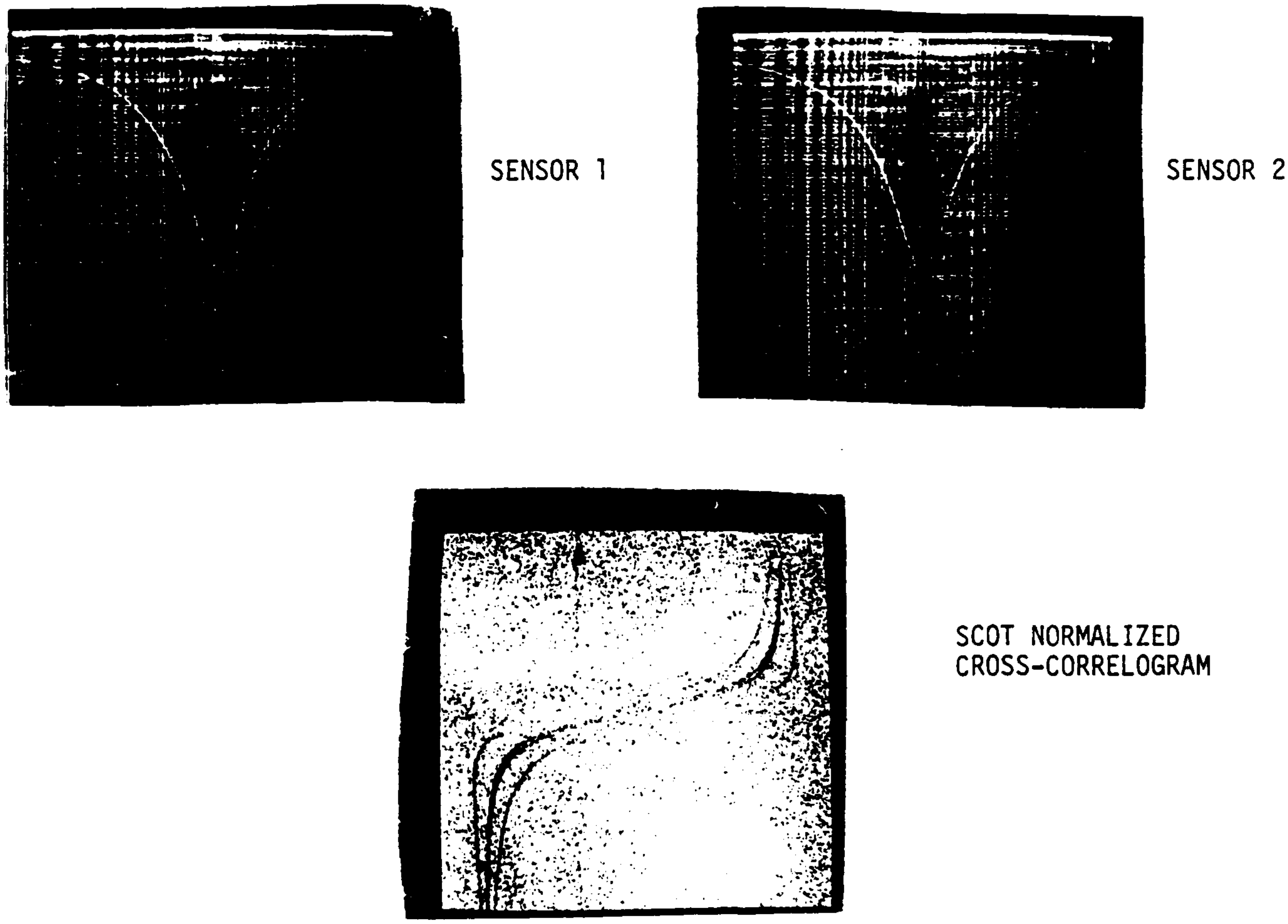


Figure A2.1 Correlograms for sensor 1, sensor 2, and the SCOT normalized correlogram between sensor 1 and sensor 2.

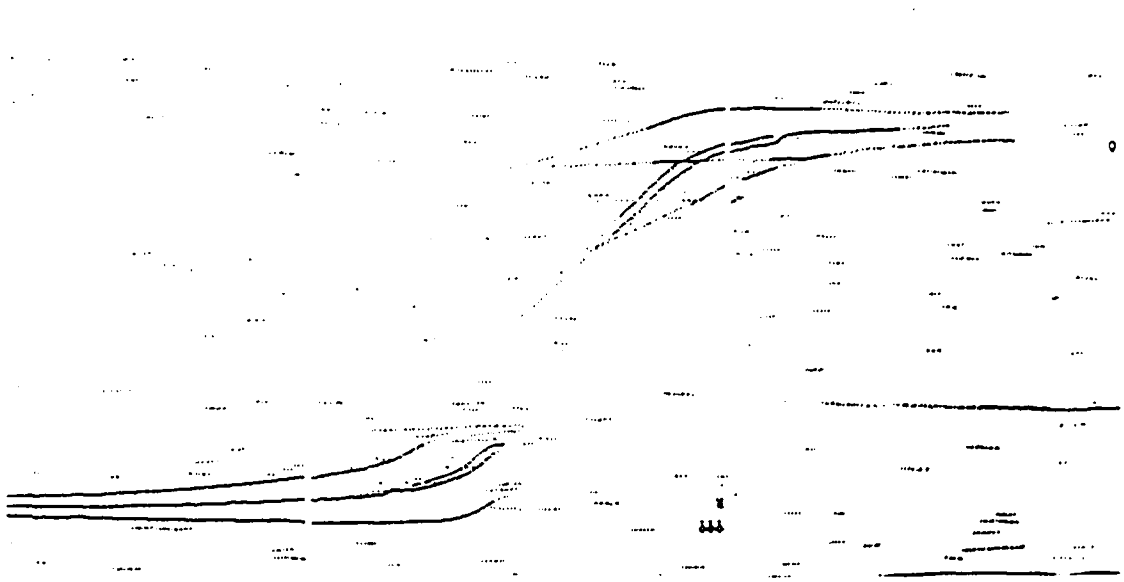


Figure A2.2 Example of the ADEC line tracking algorithm run on the SCOT normalized cross-correlogram

$$R_{C_{PA}} = \frac{2Z_sZ_T}{CD(0)} - \frac{CD(0)}{2} \tag{A2.4}$$

$$V = \frac{2X_s}{\cos\theta} \cdot \frac{1}{t_{R_{CPA_1}} - t_{R_{CPA_2}}} \tag{A2.5}$$

$$Y_c = \left[X_s^2\left[\left(\frac{2V\cos\theta}{C\left.\frac{\partial D_{12}}{\partial t}\right|_{t:D_{12}(t)=0}}\right)^2 - 1\right] - (Z_T - Z_S)^2\right]^{-1/2} \tag{A2.6}$$

Using (A2.2), one finds,

$$\cos\theta \sim 1,\ \theta \sim 0 \tag{A2.7}$$

Noting that the values of $D_1(0)$ and $D_2(0)$ are very close, and are related to R_{CPA} by (A2.4), the CPA ranges at sensors 1 and 2 will be similar, confirming the 0° bearing angle estimate.

If a 0° bearing angle is assumed, (A2.2) gives

$$C \sim 1450 \text{ m/s} \tag{A2.8}$$

with knowledge of C and θ, V can be estimated from (A2.5),

$$V \sim 7.7 \text{ m/s} = 14 \text{ KTS} \tag{A2.9}$$

Assuming Z_T is constant, the two estimates of $\hat{\rho}$ can be combined giving, from (A2.3),

$$Z_T \sim 92\text{m} \tag{A2.10}$$

Using the above value of Z_T, R_{CPA} for each of the sensors can be calculated with (A2.4)

$$R_{CPA_1} \sim 455 \text{ m} \tag{A2.11}$$

$$R_{CPA_2} \sim 470 \text{ m}$$

The Y-axis crossing can be estimated using (A2.6),

$$Y_c \sim 510 \text{ m} \tag{A2.12}$$

Note that since $\theta \sim 0$, $Y_c \sim R_{CPA_{12}}$, and therefore the value of Y_c is consistent with the values of R_{CPA_1} and R_{CPA_2}.

From this real data set, a group of self-consistent track parameter values have been obtained using various easily measured quantities. Note that in this case, the value of $\theta \sim 0$ made it difficult to determine X_c (through (55)). Consequently, additional estimates of θ, V and R_{CPA} were not obtainable.

APPENDIX 3

ACCURACY OF DEPTH ESTIMATION IN A MULTIPATH ENVIRONMENT

ACCURACY OF DEPTH ESTIMATION IN A MULTIPATH ENVIRONMENT

B. Friedlander and J.O. Smith

Systems Control Technology, Inc.
1801 Page Mill Road
Palo Alto, California 94304

ABSTRACT

The problem of estimating the depth of an underwater source from acoustic measurements is considered. The Cramer-Rao bound on the variance of the depth estimate is evaluated for the cases of one and two sensors. The effect of multipath on estimation accuracy is investigated.

This work was supported by the Office of Naval Research under Contract No. N00014-84-C-0408.

1. INTRODUCTION

The problem of estimating the depth of an underwater source is of considerable interest in various surveillance applications. Given measurements from a vertical acoustic array it is possible to estimate the angle of arrival of the signal. From this estimate and knowledge of the distance to the source it is possible to estimate its depth. If the source range is not known it can be estimated from the curvature of the wavefront at the array.

Multipath propagation is a common effect in the transmission of acoustic waves in water. The presence of multipath propagation is usually considered to be an undesirable effect which complicates the processing of the acoustic signals in localization and detection problems. However, in some circumstances the presence of multipath can be quite useful.

As an example consider the case of depth estimation when only a single sensor is available. In the absence of multipath the measurements provided by the sensor contain no information about the location of the source. However, if multipath is present, the delay between the signals propagating in the direct and the secondary paths contain information about the depth of the source. In the presence of multipath it is, therefore, possible to estimate source depth, while in its absence this is not possible.

In this paper we study the accuracy of source depth estimation both in the presence and in the absence of multipath, for the case of one and two sensors. These results can be extended in a straightforward manner to an arbitrary number of sensors.

The Cramer-Rao bound (CRB) is used to evaluate estimation accuracy. The CRB specifies the best possible accuracy achievable by any unbiased estimator [1]. In the case of stationary Gaussian processes, the CRB can be evaluated by a simple frequency domain formula as the number of observations becomes large [2]. This simplified formula has been applied to the estimation of inter-sensor time delay, and single-sensor multipath delay [3]. The purpose of this study is to extend previous work to include multipath effects in the

two-sensor case. The main question is how much better can depth be estimated when both multipath and inter-sensor time delay information are used by the estimator.

Four cases are defined to show a progression from the single-sensor case to the two-sensor case:

(1) Single-sensor case with multipath. The multipath delay is used to estimate depth.

(2) Two-sensor case, no multipath. The inter-sensor time delay is used to estimate depth.

(3) Two-sensor case with multipath. In this case both inter-sensor delay and multipath information are used to estimate depth.

(4) Two-sensor case with multipath and unknown range. When range is not known, the combination of inter-sensor delay and multipath delays suffice to determine both range and depth. The CRB for this case is also presented.

Examples are provided in each case to illustrate the performance bounds as function of system parameters.

2. ESTIMATING DEPTH FROM SINGLE-SENSOR MULTIPATH INFORMATION

The single-sensor multipath case is depicted in Figure 1. The received signal s(t) can be expressed as

$$s(t) = x(t) + gx(t-D_1) + e(t) \tag{1}$$

where x(t) is the signal received by direct propagation from the source to the sensor, e(t) is uncorrelated measurement noise, and $gx(t-D_1)$ is the attenuated and delayed version of the source signal arising from a surface bounce. The corresponding power spectral density (PSD) function at the sensor is given by

$$S_s(\omega) = S_x(\omega)\left|1 + ge^{-j\omega D_1}\right|^2 + S_e(\omega)$$

$$= S_x(\omega)[(1+g^2) + 2g\cos(\omega D_1)] + S_e(\omega), \tag{2}$$

for ω in the range $[-W,W]$, where $W = 2\pi B$, and B is the bandwidth in Hz.

In terms of the geometry (see figure 1), the multipath delay is given by

$$D_1 = \frac{1}{c}\{[y^2 + (z+z_1)^2]^{1/2} - [y^2 + (z-z_1)^2]^{1/2}\}, \tag{3}$$

where c is the speed of sound. Thus, assuming the range y and sensor-depth z_1 are known, the source-depth z is determined by the multipath delay D_1.

Using Whittle's formula [2], the CRB for the variance of the depth estimate is given by

$$\left[\frac{N}{2}\cdot\frac{1}{2W}\int_{-W}^{W}\left[\frac{\frac{\partial S(\omega)}{\partial D_1}}{S(\omega)}\right]^2 d\omega\left(\frac{\partial D_1}{\partial z}\right)^2\right]^{-1}$$

$$= \left[\frac{N}{2}\cdot\frac{1}{2W}\int_{-W}^{W}\frac{[-2g\omega\sin\omega D_1]^2}{S^2(\omega)}d\omega\right]^{-1}\left(\frac{\partial z}{\partial D_1}\right)^2$$

$$= CRB(D_1)\left(\frac{\partial z}{\partial D_1}\right)^2 , \qquad (4)$$

where N is the number of independent observation intervals. (In continuous time, the observation time is defined as T=N/2B. In discrete time, when B equals half the sampling rate, N is the number of sampled observations.) Equation (4) can be interpreted as a generalized form of the CRB, see [4].

2.1 SPECIAL CASES

When the PSD of x(t) and e(t) are constant for $|\omega| \leq W$, and zero for $|\omega| > W$, with the signal-to-noise-ratio (SNR) defined as S_x/S_e, we have the following approximations (see [3],[5]):

(i) $WD_1 >> 1$, $SNR << 1$

$$CRB(D_1) = \frac{3\pi}{TW^3 g^2} \frac{1}{SNR^2} = \frac{3}{NW^2 g^2} \frac{1}{SNR^2} . \qquad (5)$$

(ii) $WD_1 >> 1$, $SNR >> 1$

$$CRB(D_1) = \frac{3\pi(1-g^2)}{TW^3 g^2} , \qquad (6)$$

which is independent of SNR.

(iii) $WD_1 << 1$

$$CRB(D_1) = \frac{3\pi}{TW^3} \frac{[(1+g)^2 SNR+1]}{g^2 SNR^2} \cdot \frac{1}{1.2W^2 D_1^4} . \qquad (7)$$

Note that the CRB depends on the multipath delay D_1 for small delay-bandwidth product WD_1, but not for $WD_1 >> 1$.

When the range to the source is large compared to the depths of the source and sensor, the derivative $\frac{\partial z}{\partial D_1}$ has a simple form:

$$\frac{\partial z}{\partial D_1} = \frac{cy}{\partial z} , \qquad z, z_1 << y . \qquad (8)$$

Note that the variance of the depth estimate is inversly proportional to

$(2z/y)^2$, where $(2z/y)$ is approximately the angle between the source and its "reflection", as viewed from the sensor.

2.2 EXAMPLES

Figure 2 shows a specific scenario for which the CRB was evaluated. Figures 3, 4, and 5 display the normalized Cramer-Rao bound, $CRB(z)/z^2$ as a function of SNR, for delay-bandwidth products BD_1 = 0.1, 1, and 10, respectively. For the geometry in Figure 2, these time-bandwidth products correspond to bandwidths B = 3.78, 37.78, and 377.78, respectively. Each figure shows three different values of g: g = -0.1, -0.5, and -0.90. As expected, increased reflection magnitude $|g|$ improves the ability to estimate z, as does increased delay-bandwidth product.

3. ESTIMATING DEPTH FROM INTER-SENSOR DELAY

Figure 6 shows the general case of two sensors configured in a vertical array at depths z_1 and z_2, respectively. In this section multipath is assumed absent. The inter-sensor delay, i.e., the time-difference of arrival (TDOA) between sensors 1 and 2, is defined as

$$D_{12} = \frac{1}{c}\{[y^2 + (z-z_1)^2]^{1/2} - [y^2 + (z-z_2)^2]^{1/2} . \tag{9}$$

The respective received signals are given by (analogous to (1)),

$$s_1(t) = x(t) + e_1(t) ,$$

$$s_2(t) = x(t-D_{12}) + e_2(t) . \tag{11}$$

where $e_1(t)$ and $e_2(t)$ are assumed to be uncorrelated. The power spectral density of the 2-vector process $s(t) = [s_1(t), s_2(t)]^T$ is equal to

$$S(\omega) = \begin{bmatrix} S_x(\omega) + S_{e_1}(\omega) & S_x(\omega)e^{j\omega D_{12}} \\ S_x(\omega)e^{-j\omega D_{12}} & S_x(\omega) + S_{e_2}(\omega) \end{bmatrix} . \tag{12}$$

For the vector case, the asymptotic form of the CRB for the covariance of the parameter estimates θ becomes [2],

$$\mathrm{Cov}(\theta) \geq J^{-1} \quad , \quad J = [J_{ij}] ,$$

$$J_{ij} = \frac{T}{4\pi} \int_{-W}^{W} \mathrm{tr}\{S^{-1}(\omega) \frac{\partial S(\omega)}{\partial \theta_i} S^{-1}(\omega) \frac{\partial S(\omega)}{\partial \theta_j} \} d\omega , \tag{13}$$

where tr(A) denotes the trace of the matrix A. The CRB for D_{12} is then given by

$$\mathrm{CRB}(D_{12}) = \left[\frac{T}{4\pi} \int_{-W}^{W} \mathrm{tr} \left[S^{-1}(\omega) \frac{\partial S(\omega)}{\partial D_{12}} \right]^2 d\omega \right]^{-1}$$

$$= \left[\frac{T}{2\pi} \int_{-W}^{W} \frac{\omega^2 \mathrm{SNR}^2}{1+2\mathrm{SNR}} d\omega \right]^{-1} , \tag{14}$$

and the CRB for depth z is given by

$$\mathrm{CRB}(z) = \mathrm{CRB}(D_{12}) \left(\frac{\partial z}{\partial D_{12}} \right)^2 . \tag{15}$$

3.1 SPECIAL CASES

If the noise and signal PSD functions are assumed flat in [-W,W] and zero elsewhere, then

$$\mathrm{CRB}(D_{12}) = \frac{3\pi}{W^3 T} \frac{1+2\mathrm{SNR}}{\mathrm{SNR}^2} . \tag{16}$$

For SNR << 1 and $WD_{12} \gg 1$, we have (comparing with (5))

$$\frac{\mathrm{CRB}(D_{12})}{\mathrm{CRB}(D_1)} = g^2 . \tag{17}$$

At long ranges, $y \gg z, z_1, z_2$, we have (cf. (8))

$$\frac{\partial z}{\partial D_{12}} = \frac{cy}{z_2 - z_1} \tag{18}$$

Thus,

$$\frac{\mathrm{CRB}(z)[\text{two-sensors, no multipath}]}{\mathrm{CRB}(z)[\text{one-sensor, multipath-only}]} = (2z)^2 \cdot \frac{g^2}{(z_2 - z_1)^2} \tag{19}$$

When g=0 or z=0, no depth information is obtained in the single-sensor multipath case, while for $z_1 = z_2$, no depth information is available in the TDOA case. These facts are, of course, to be expected.

3.2 EXAMPLES

Figure 7 shows the normalized CRB, $CRB(z)/z^2$, as a function of SNR for three different signal bandwidths B = 3.78, 37.78, and 377.78. The geometry is the same as in the single-sensor case (figure 2) with the second sensor being located at $z_2 = 400$. The three curves may be compared with the results in figures 3, 4, and 5, respectively. Figure 8 shows the effect of varying the geometry, using B = 377.8 Hz and $z_2 = 220$, 400, and 600.

4. ESTIMATING DEPTH FROM BOTH MULTIPATH AND INTER-SENSOR DELAYS

Figure 9 shows the geometry for the two-sensor case in which both multipath and inter-sensor delays are to be estimated. The two multipath delays are given by

$$D_1 = \frac{1}{c}\{[y^2+(z+z_1)^2]^{1/2} - [y^2+(z-z_1)^2]^{1/2}\} ,$$

$$D_2 = \frac{1}{c}\{[y^2+(z+z_2)^2]^{1/2} - [y^2+(z-z_2)^2]^{1/2}\} , \tag{20}$$

and the TDOA is again equal to (cf. (10))

$$D_{12} = \frac{1}{c}\{[y^2+(z-z_1)^2]^{1/2} - [y^2+(z-z_2)^2]^{1/2}\} . \tag{21}$$

The received signals at sensors 1 and 2 are given by

$$y_1(t) = x(t-\tau_{11}) + gx(t-\tau_{21}) + e_1(t) ,$$

$$y_2(t) = x(t-\tau_{12}) + gx(t-\tau_{22}) + e_2(t) , \tag{22}$$

where

$$\begin{aligned} D_1 &= \tau_{21} - \tau_{11} , \\ D_2 &= \tau_{12} - \tau_{22} , \\ D_{12} &= \tau_{11} - \tau_{12} . \end{aligned} \tag{23}$$

The PSD matrix is (cf. (2) and (12))

$$S(\omega) = \begin{bmatrix} S_x(1+g^2+2g\cos\omega D_1) & S_x e^{-j\omega D_{12}}(1+ge^{-j\omega D_1})(1+ge^{-j\omega D_2}) \\ S_x e^{j\omega D_{12}}(1+ge^{j\omega D_1})(1+ge^{j\omega D_2}) & S_x(1+g^2+2g\cos\omega D_2) \end{bmatrix} . \tag{24}$$

The CRB for depth estimation is then

$$\mathrm{CRB}(z) = \left[\frac{T}{4\pi} \int_{-W}^{W} J(\omega)\,d\omega \right]^{-1} , \qquad (25)$$

where

$$J(\omega) = [S^{-1}(\omega)\frac{\partial S(\omega)}{\partial z}]^2 ,$$

$$S(\omega) = \begin{bmatrix} S_1(\omega) & S_{12}(\omega) \\ S_{12}^*(\omega) & S_2(\omega) \end{bmatrix},$$

$$\frac{\partial S(\omega)}{\partial z} \triangleq \dot{S}(\omega) = \begin{bmatrix} \dot{S}_1(\omega) & \dot{S}_{12}(\omega) \\ \dot{S}_{12}^*(\omega) & \dot{S}_2(\omega) \end{bmatrix}, \qquad (26)$$

and

$$S_1(\omega) = S_x(\omega)[1+g_1^2+2g_1\cos\omega D_1] + S_{e_1}(\omega) ,$$

$$S_2(\omega) = S_x(\omega)[1+g_2^2+2g_2\cos\omega D_2] + S_{e_2}(\omega) ,$$

$$S_{12}(\omega) = S_x(\omega)e^{-j\omega D_{12}}(1+g_1e^{-j\omega D_1})(1+g_2e^{-j\omega D_2}) , \qquad (27)$$

$$\dot{S}_1(\omega) = \frac{\partial S_1(\omega)}{\partial z} = 2g_1S_x\omega \sin(\omega D_1)\left[\frac{z+z_1}{\tau_{21}} - \frac{z-z_1}{\tau_{11}}\right]\frac{1}{c^2} ,$$

$$\dot{S}_2(\omega) = \frac{\partial S_2(\omega)}{\partial z} = 2g_2S_x\omega \sin(\omega D_2)\left[\frac{z+z_2}{\tau_{22}} - \frac{z-z_2}{\tau_{12}}\right]\frac{1}{c^2} ,$$

$$\dot{S}_{12}(\omega) = \frac{\partial S_{12}(\omega)}{\partial z} = j\omega S_x e^{-j\omega D_{12}}$$

$$\cdot \left[(1+g_1e^{-j\omega D_1})\left(\frac{z-z_2}{\tau_{12}}\right) + g_2e^{-j\omega D_2}\left(1+g_1e^{-j\omega D_1}\right)\left(\frac{z+z_2}{\tau_{22}}\right)\right.$$

$$-\left(1+g_2 e^{-j\omega D_2}\right)\left(\frac{z-z_1}{\tau_{11}}\right) - g_1 e^{-j\omega D_1}\left(1+g_2 e^{-j\omega D_2}\right)\left(\frac{z+z_1}{\tau_{21}}\right)\frac{1}{c^2}\Bigg] . \qquad (28)$$

4.1 EXAMPLES

(i) Different Bandwidths, B=3.78, 37.78, 377.78

Figures 10, 11, and 12 show the normalizede CRB, $CRB(z)/z^2$, for the three delay-bandwidth cases seen in the previous examples. The case of g=0 in each figure is precisely the CRB curve obtained in the previous section for TDOA based depth estimation (no multipath). Note that use of multipath delay estimates can improve the relative variance in the depth estimate by 10 to 15 dB when the multipath is strong.

(ii) Changing Geometry

Figures 13, 14, 12, 15 show a sequence of CRB curves in which the second sensor is taken deeper and deeper, i.e., z_2 = 200, 220, 400, 600 . All are at the large bandwidth B = 377.78. In figure 13, the g=0 curve is at infinity because for $z_1 = z_2$ and g = 0 there is no TDOA or multipath information on which to base a depth estimate. We find that increasing the depth of sensor 2 improves the bound. There is more improvement near g=0 than near g=-0.9 indicating that it is the TDOA component that is benefiting from increased z_2 . This is reasonable in view of equations (8) and (18) which show that at long range, the derivative of z with respect to TDOA depends strongly on z_2-z_1 while the sensitivity to multipath delay does not.

5. ESTIMATING DEPTH AND RANGE FROM MULTIPATH AND INTER-SENSOR DELAYS

In the single-sensor case, it has been shown [6] that, assuming the range error is uncorrelated with the multipath delay error, the variance of the depth estimate becomes a sum of two terms involving the respective variances of range and delay:

$$\mathrm{Var}(z) = \left(\frac{\partial z}{\partial D_1}\right)^2 \mathrm{Var}(D_1) + \left(\frac{\partial z}{\partial y}\right)^2 \mathrm{Var}(y) \ , \tag{29}$$

where the sensitivity of y to z is defined for fixed D_1 :

$$\frac{\partial z}{\partial y} = \frac{cD_1}{2z_1} \tag{30}$$

At long range $(z,\ z_1 << y)$, we may use (8) to obtain

$$\frac{\mathrm{Var}(z)}{z^2} = \frac{\mathrm{Var}(D_1)}{D_1^2} + \frac{\mathrm{Var}(y)}{y^2} \tag{31}$$

For two sensors with no multipath (TDOA only), a similar situation is obtained.

In the case of two sensors with multipath it is possible to estimate both range and depth. An interesting question is the following: how much is depth estimation accuracy effected by the lack of knowledge of the range? To answer this question we evaluate the CRB on the covariance matrix of the estimate of the parameter vector $[z,y]^T$. Using Whittee's formula we note that

$$\mathrm{CRB}\{[z,y]^T\} = \begin{bmatrix} J_{zz} & J_{zy} \\ J_{yz} & J_{yy} \end{bmatrix}^{-1} \ , \tag{32}$$

where (cf. (25)),

$$J_{zz} = \frac{T}{4\pi} \int_{-\infty}^{\infty} [S(\omega)^{-1} \frac{\partial S(\omega)}{\partial z}][S(\omega)^{-1} \frac{\partial S(\omega)}{\partial z}]\, d\omega \ , \tag{33a}$$

and

$$J_{zy} = J_{yz}^{*} = \frac{T}{4\pi}\int_{-\infty}^{\infty}[S(\omega)^{-1}\frac{\partial S(\omega)}{\partial z}][S(\omega)^{-1}\frac{\partial S(\omega)}{\partial y}]d\omega \ , \tag{33b}$$

$$J_{yy} = \frac{T}{4\pi}\int_{-\infty}^{\infty}[S(\omega)^{-1}\frac{\partial S(\omega)}{\partial y}][S(\omega)^{-1}\frac{\partial S(\omega)}{\partial y}]d\omega \ . \tag{33c}$$

The matrices $S(\omega)$ and $\frac{\partial S(\omega)}{\partial z}$ are defined in equations (25)-(26), while

$$\frac{\partial S(\omega)}{\partial y} = \begin{bmatrix} \frac{\partial S_1(\omega)}{\partial y} & \frac{\partial S_{12}(\omega)}{\partial y} \\ \frac{\partial S_{21}(\omega)}{\partial y} & \frac{\partial S_2(\omega)}{\partial y} \end{bmatrix} \tag{34}$$

where

$$\frac{\partial S_1(\omega)}{\partial y} = 2g_2S_{x\omega}\ \sin(\omega D_1)[\frac{1}{\tau_{21}} - \frac{1}{\tau_{11}}]\ \frac{1}{c^2} \tag{35}$$

$$\frac{\partial S_2(\omega)}{\partial y} = 2g_2S_{x\omega}\ \sin(\omega D_2)[\frac{1}{\tau_{22}} - \frac{1}{\tau_{12}}]\ \frac{y}{c^2}$$

$$\frac{\partial S_{12}(\omega)}{\partial y} = \frac{\partial S_{21}^{*}(\omega)}{\partial y} = j\omega S_x e^{-j\omega D_{12}} \cdot [[(1+g_1e^{-j\omega D_1})\ \frac{1}{\tau_{12}}$$

$$+ g_2e^{-j\omega D_2}(1+g_1e^{-j\omega D_1})\ \frac{1}{\tau_{22}} - (1 + g_2e^{-j\omega D_2})\ \frac{1}{\tau_{11}}$$

$$- g_1e^{-j\omega D_1}(1+g_2e^{-j\omega D_2})\ \frac{1}{\tau_{21}}\ ]\ \frac{y}{c^2}$$

Examples

Figures 16, 17, and 18 show the normalized CRB, $CRB(z)/z^2$, for the three bandwidths considered in the previous examples (BD_1=0.1, 1, 10) . In this particular geometry, the TDOA component of the depth estimate is independent of the range due to the hyperbola of constant TDOA degenerating to a straight line. Consequently, for g=0, the CRB curves reduce to the TDOA-only case of figure 7. Note that at small BD_1 , the presence of multipath only makes the estimates worse, while at intermediate and high BD_1 , the depth estimate is improved by multipath.

Figure 19 shows the results at high BD_1 for a different geometry (z_2=600 instead of 400). In this case the TDOA component of the depth estimate depends on range. This figure compares with figure 18. In the non-degenerate geometry, the unknown range adds significantly to the uncertainty in depth.

6. DISCUSSION

The performance analysis presented in this paper shows the potential usefulness of multipath information in enhancing the accuracy of depth estimation. The results presented here are asymptotic, and need to be verified by finite data simulations. Two related issues of practical interest are:

(i) The Case of Unknown g

The earlier analysis assumed perfect knowledge of the relfection coefficient g. When g is unknown, it can be estimated from the data. In [6] we have shown for the examples considered here that the asymptotic accuracy of depth estimation is not affected by the need to estimate g, provided that $WD \gg 1$, where $D = \max\{D_1, D_2, D_{12}\}$.

(ii) The Case of Unknown SNR

Similarly to the case of unknown g, it can be shown [6] that the need to estimate SNR does not affect the asymptotic depth estimation accuracy, provided that $WD \gg 1$.

Finally, we note that these results can be extended in a straightforward manner to the case of M sensors. In this case $S(\omega)$ and $\hat{S}(\omega)$ will be MxM matrices. The entries of these matrices can be easily evaluated by obvious extrapolation from the case of M=2.

REFERENCES

1. H. Cramer, Mathematical Methods of Statistics, Princeton, N.J.: Princeton University, 1951, Ch. 22-23.

2. P. Whittle, "The Analysis of Multiple Stationary Time Series," J. Royal Statistical Society, vol. 15, pp. 125-139, 1953.

3. B. Friedlander, "On The Cramer Rao Bound for Time Delay and Doppler Estimation," IEEE Trans. Information Theory, vol. 15-30, No. 3, pp. 575-580, May 1984.

4. S. Zacks, The Theory of Statistical Inference, John Wiley & Sons, 1971.

5. J.P. Ianniello, "The Variance of Multipath Time Delay Estimation Using Autocorrelation," Tech. Memorandum TM 831008, Naval Underwater Systsems Center, New London, CT, 24 January 1983.

6. B. Friedlander and J.O. Smith, "Multipath Delay Estimation," Tech. Report 5466-04, Systems Control Technology, Inc., December 1983.

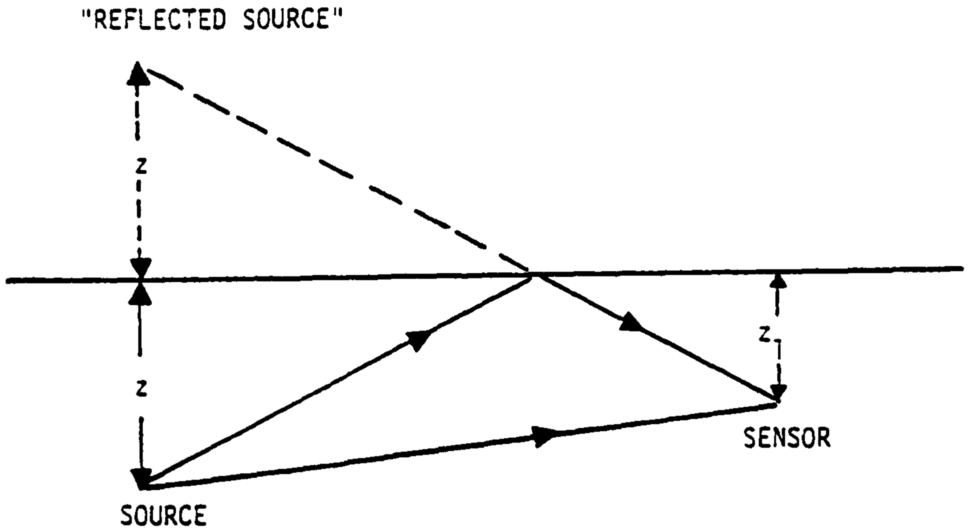


Figure 1: Single sensor geometry

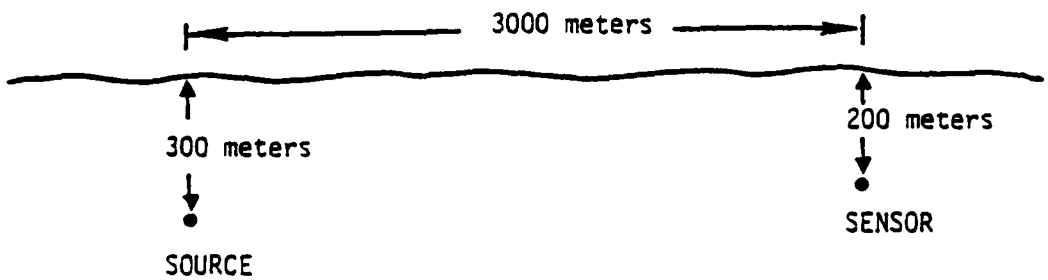


Figure 2: Single Sensor Example

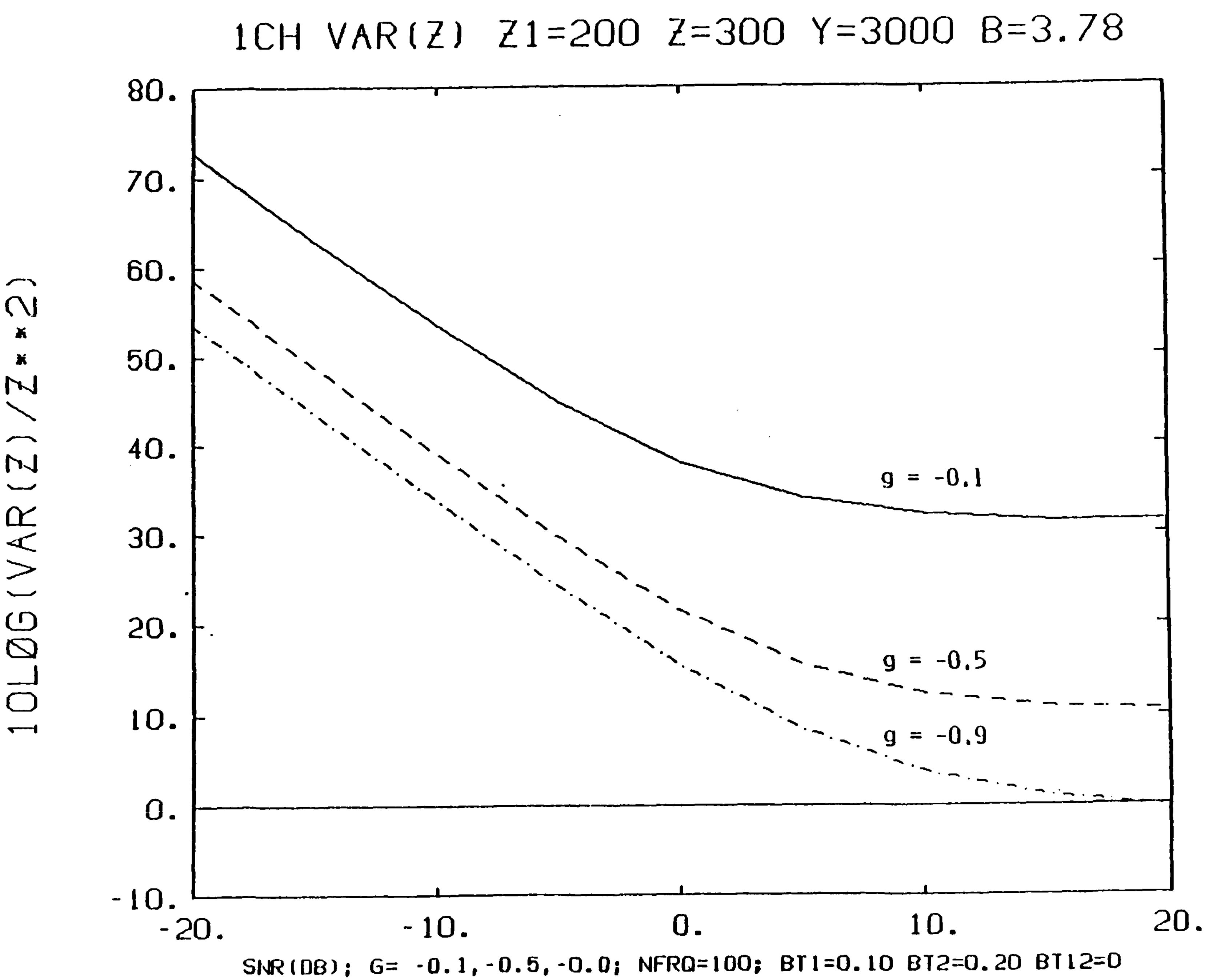


Figure 3: Single sensor with multipath, B = 3.78

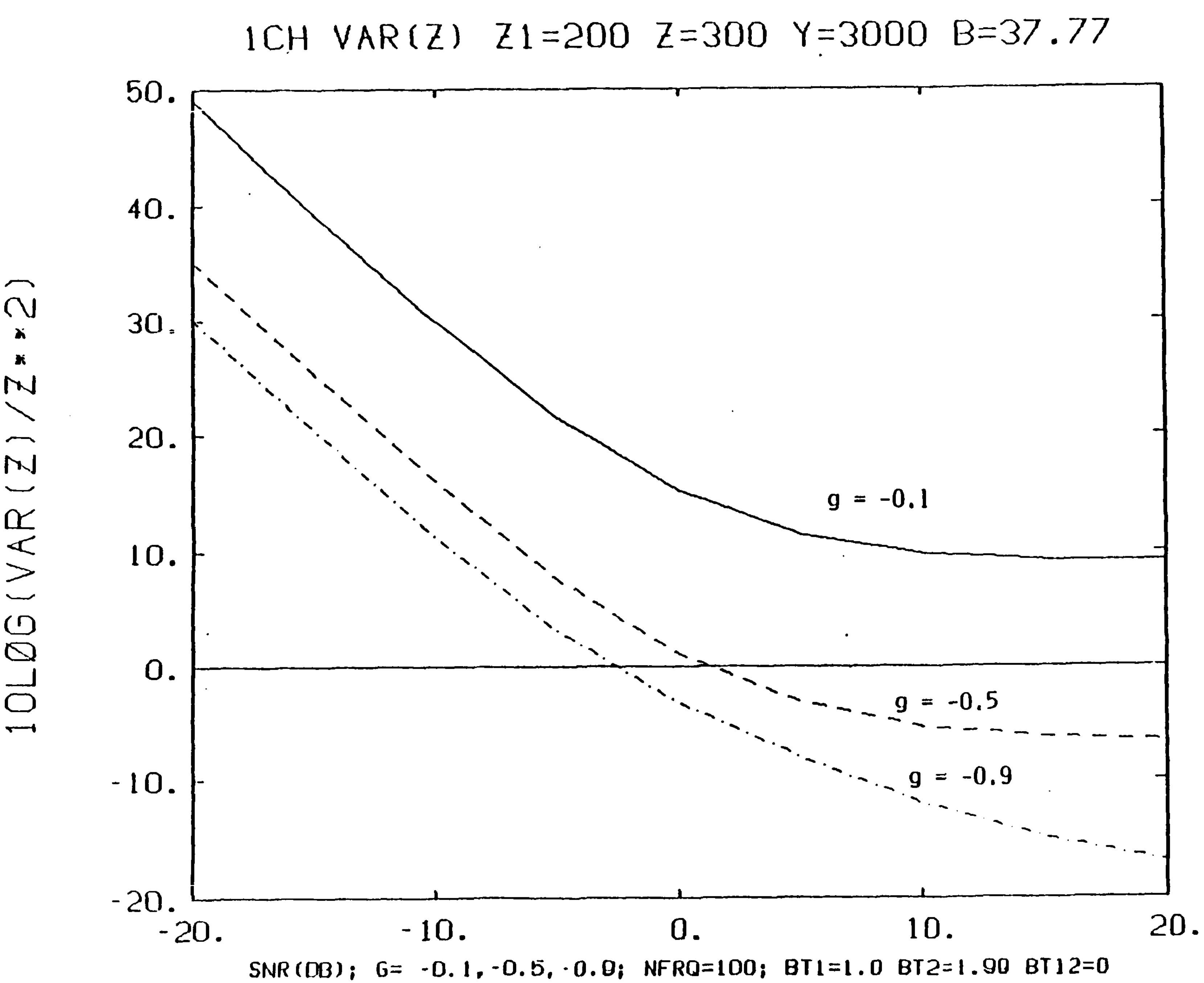


Figure 4: Single sensor with multipath, B = 37.77

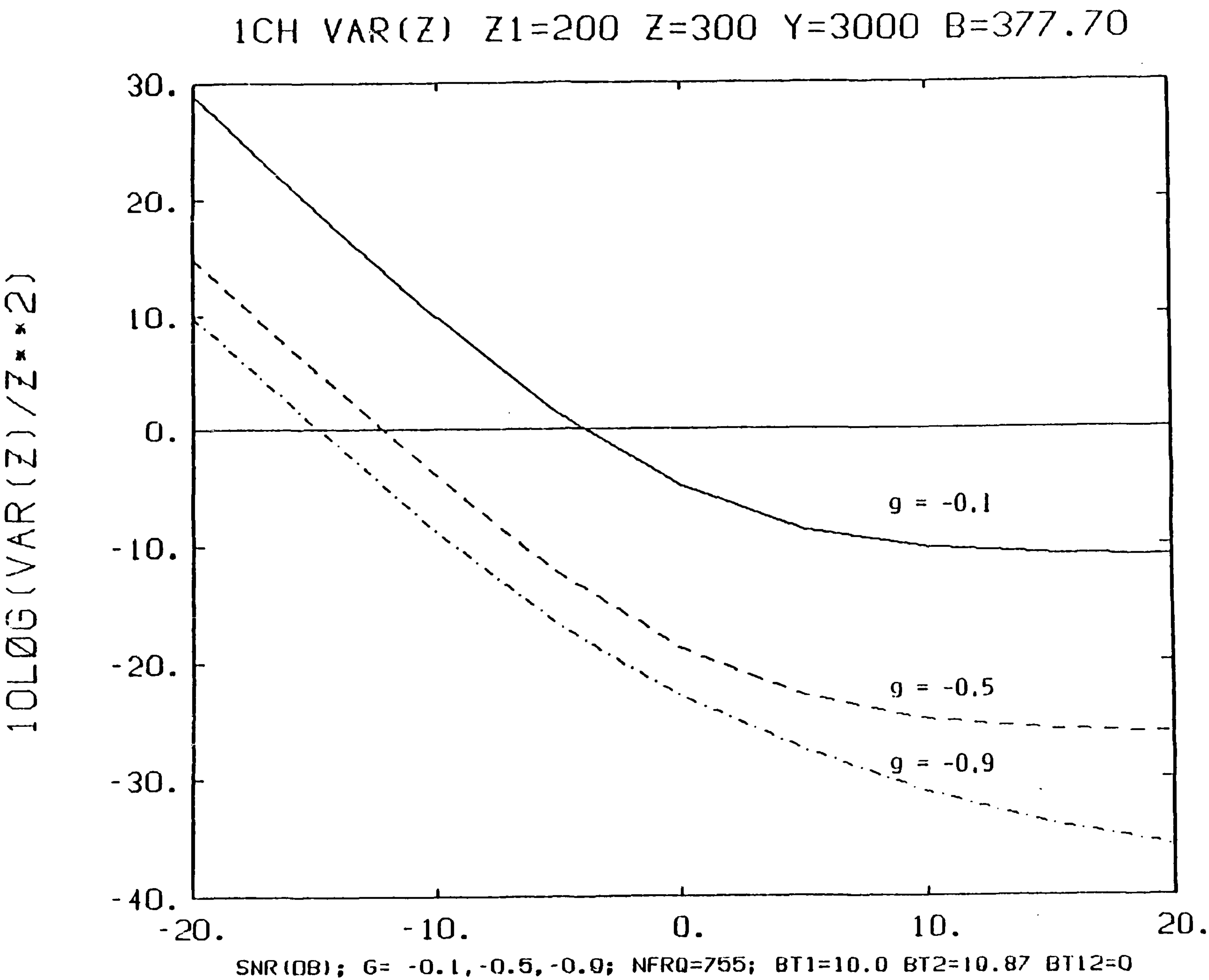
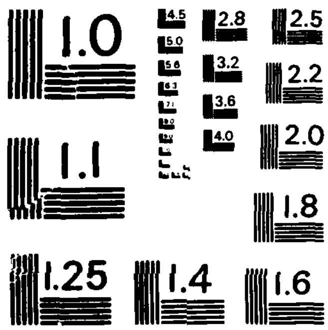


Figure 5: Single sensor with multipath, B = 377.70

AD-A166 044 TRACK PARAMETER EXTRACTION USING MULTIPATH DELAY AND DOPPLER INFORMATION(U) SYSTEMS CONTROL TECHNOLOGY INC PALO ALTO CA K LASHKARI ET AL. FEB 86 5517 2/2

UNCLASSIFIED N00014-84-C-0408 F/G 17/1 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
1.1
1.25
1.4
1.6

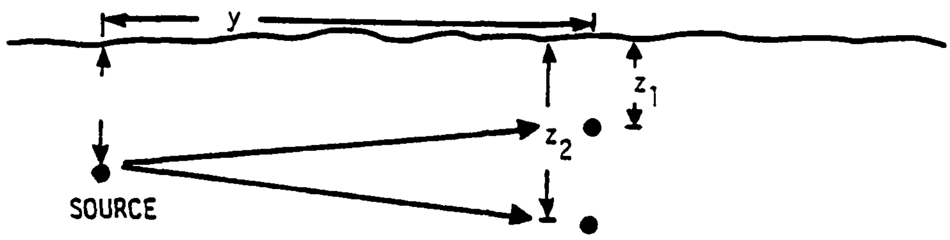


Figure 6: Geometry for two sensors no multipath

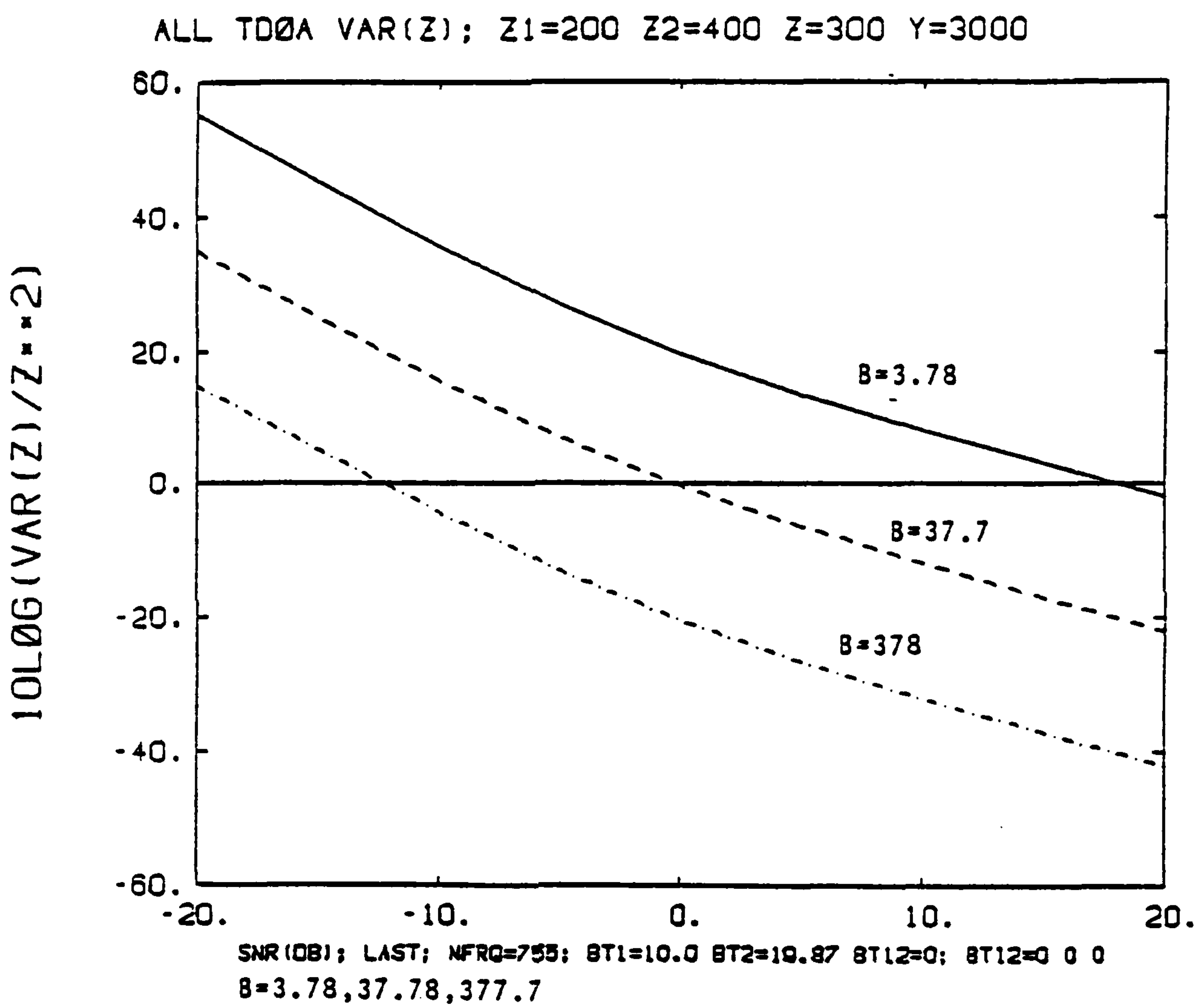


Figure 7: Two sensors no multipath example -- varying bandwidth

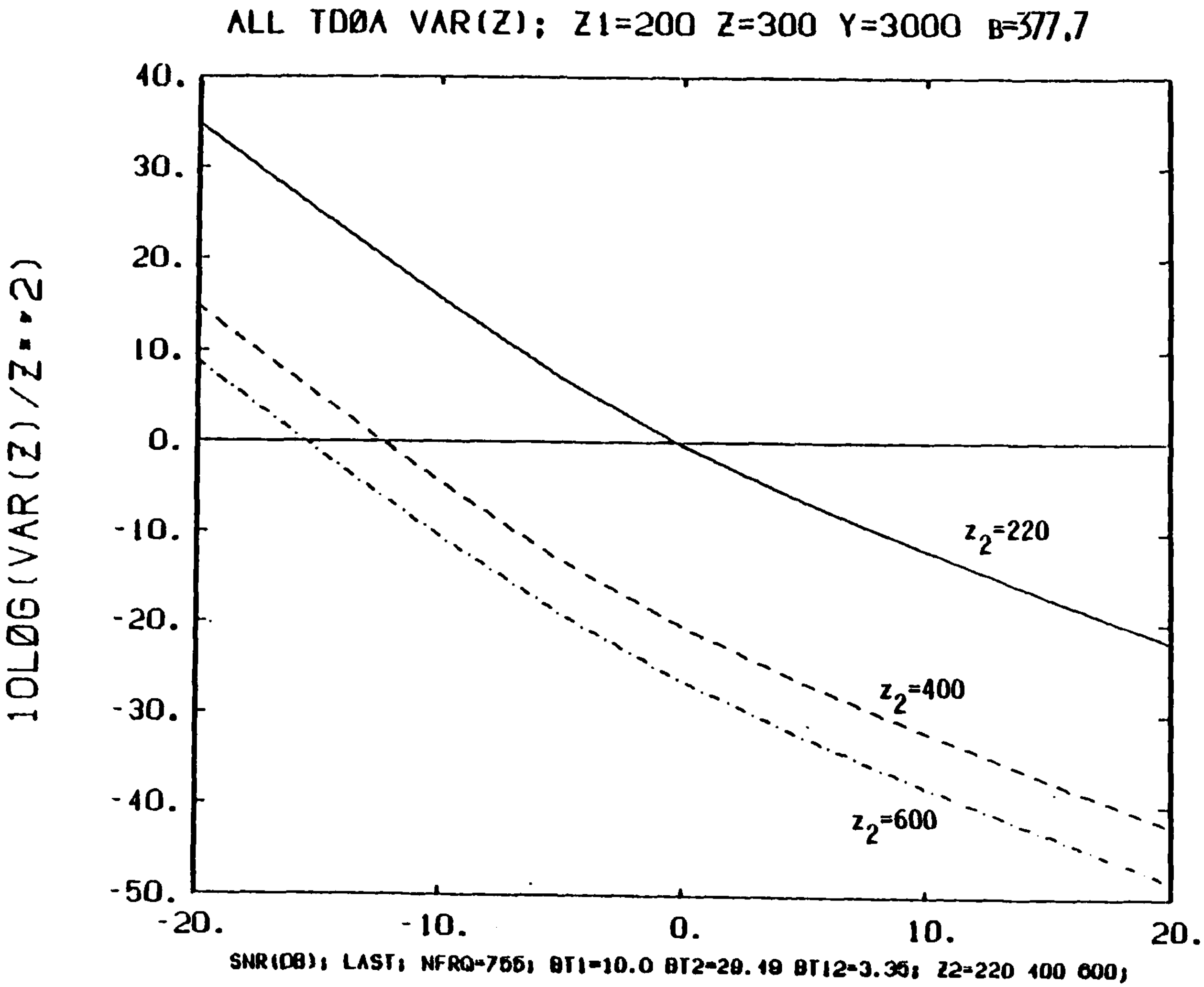


Figure 8: Two sensors no multipath example -- varying sensor separation

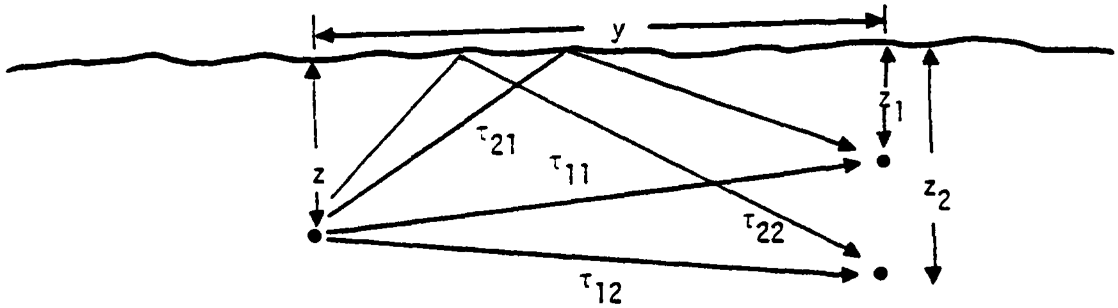


Figure 9: Geometry for two sensors with multipath

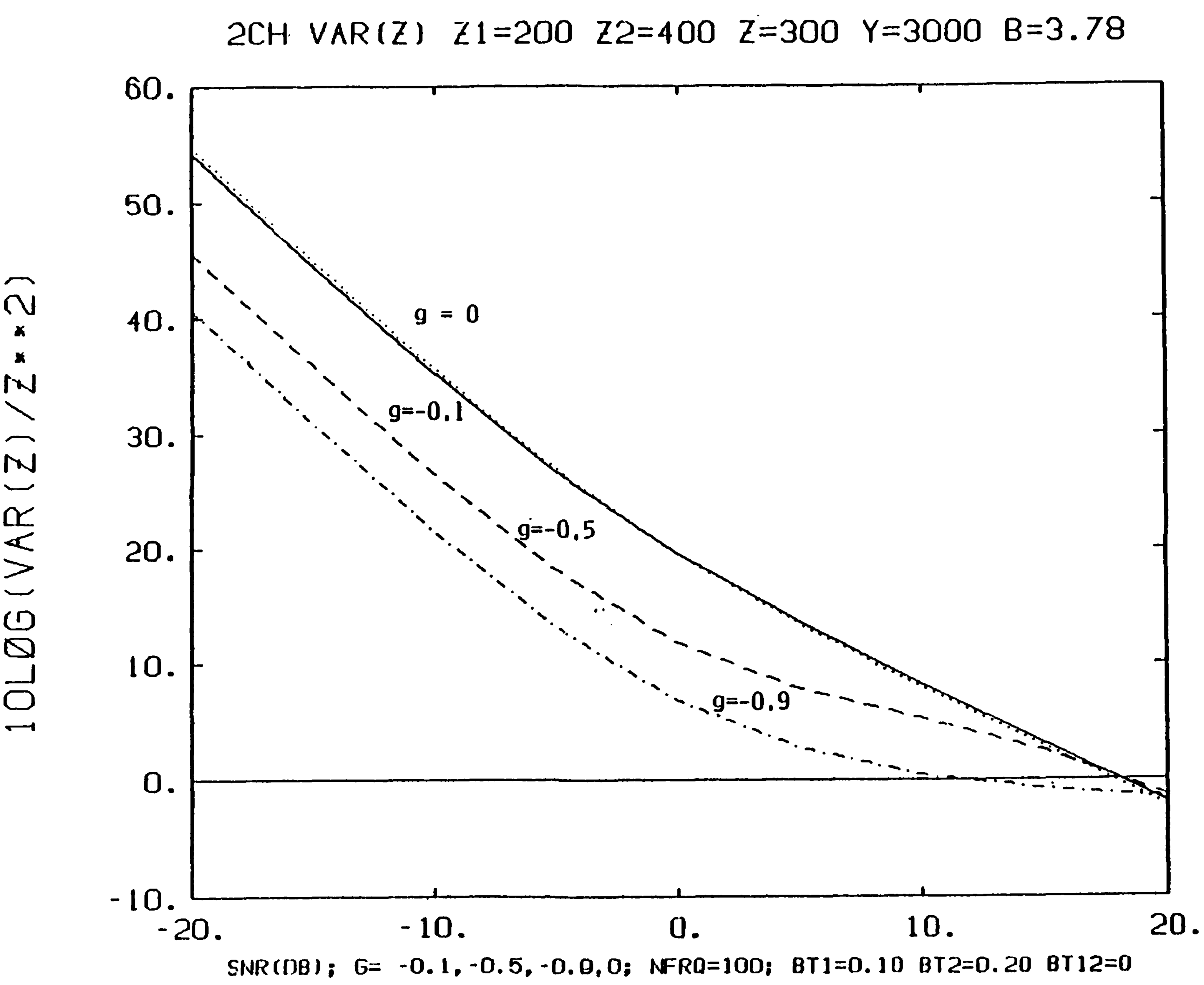


Figure 10: Two sensors with multipath -- z_2 = 400, B = 3.78

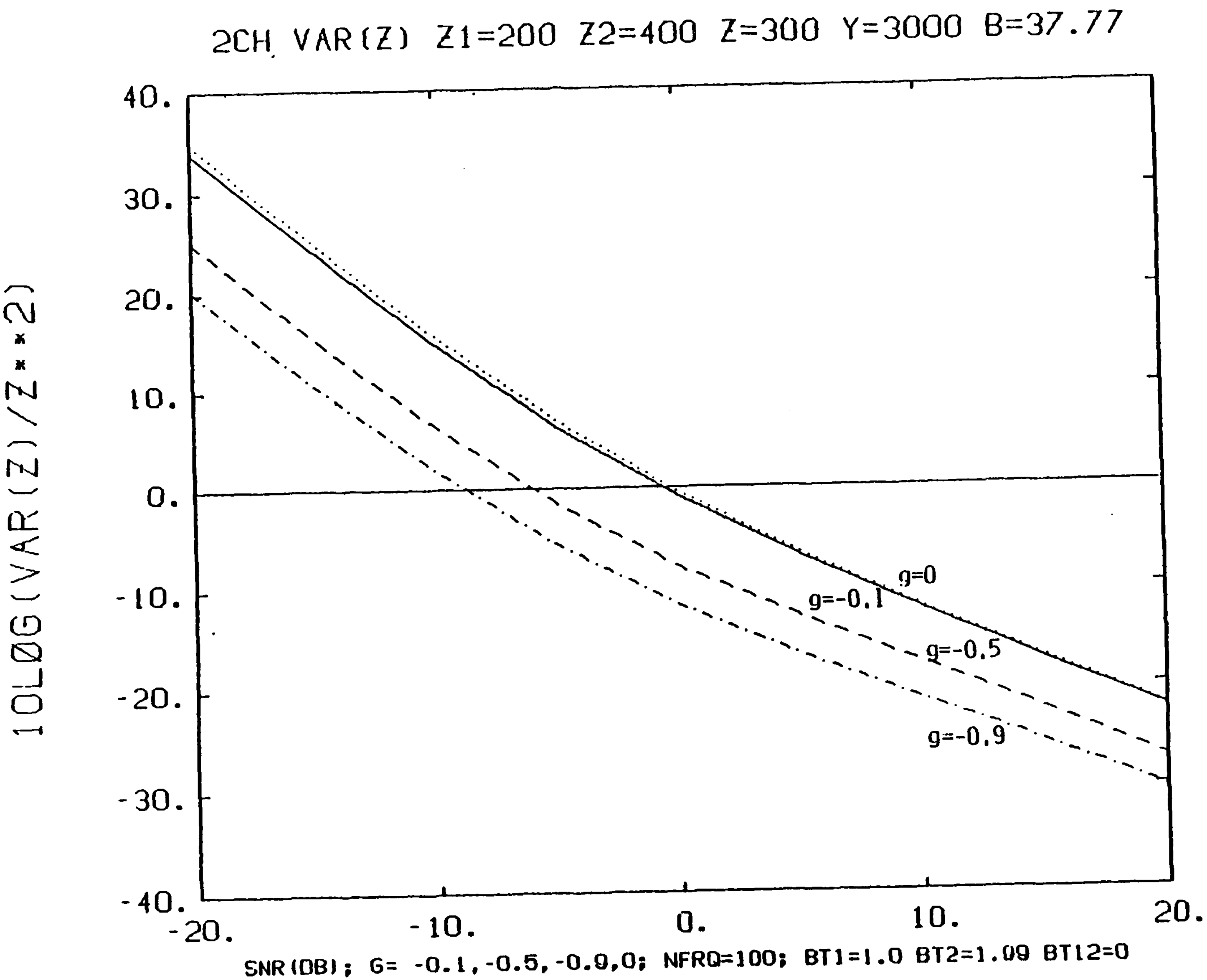


Figure 11: Two sensors with multipath -- z_2 = 400, B = 37.77

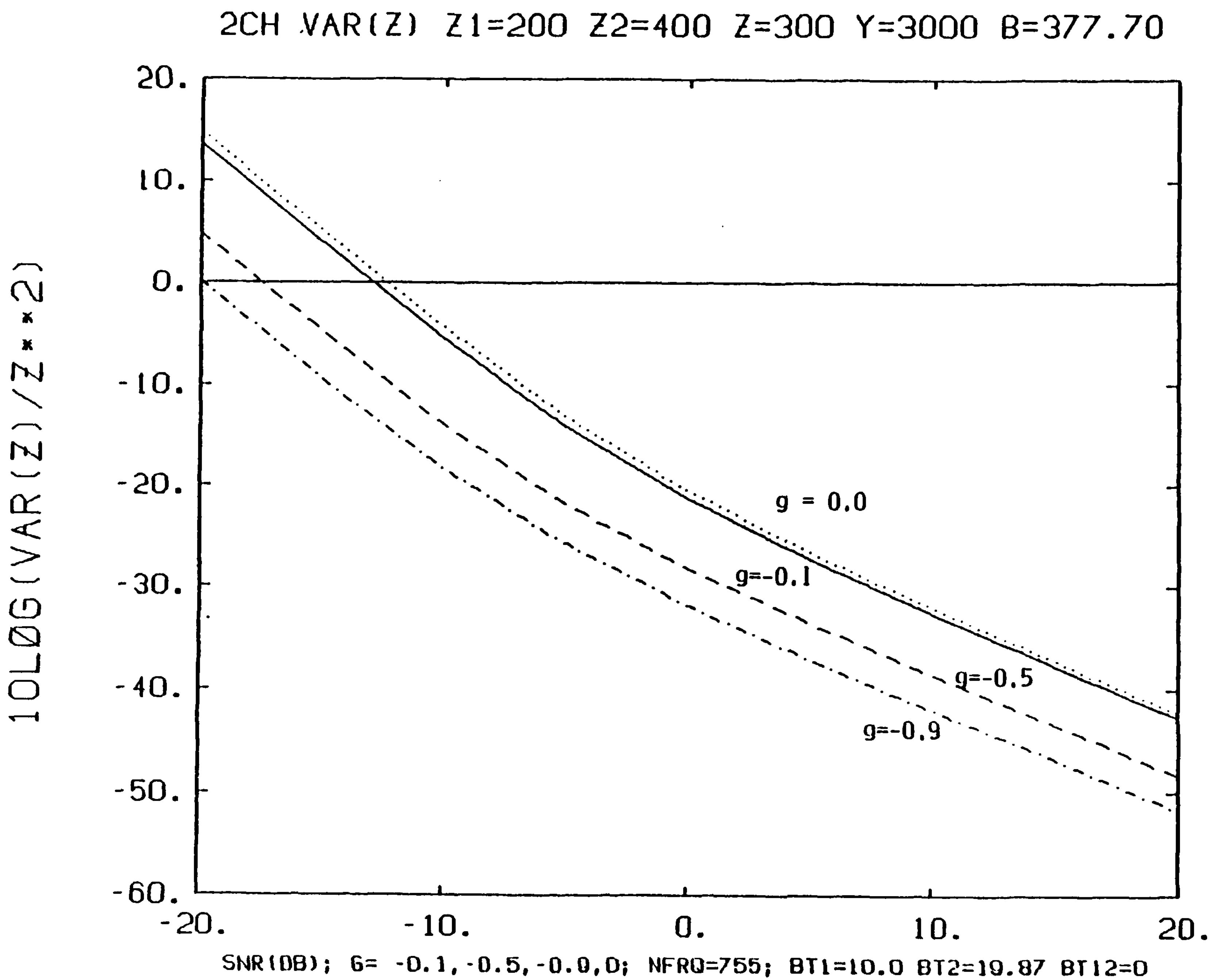


Figure 12: Two sensors with multipath -- z_2 = 400, B = 377.70

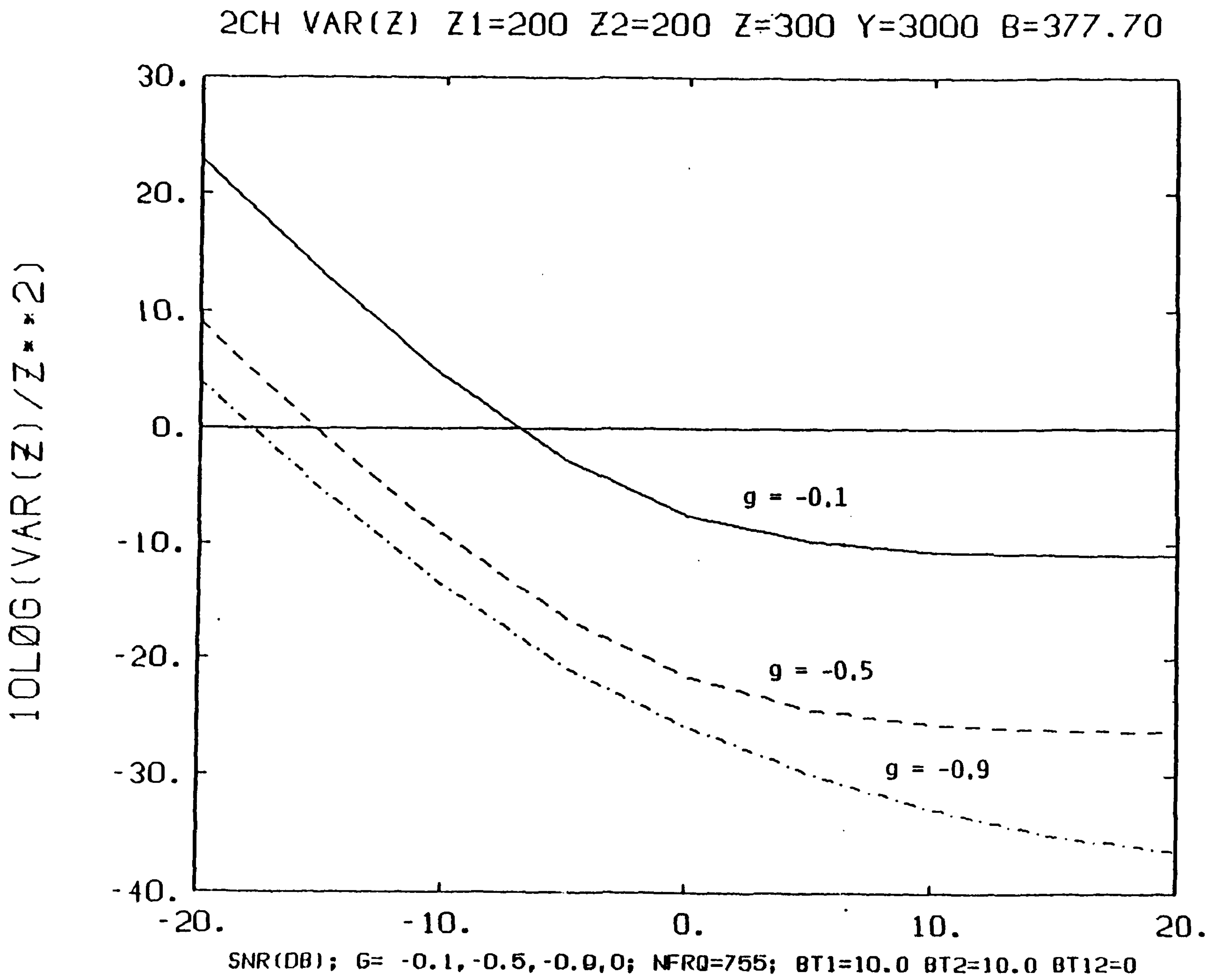


Figure 13: Two sensors with multipath -- z_2 = 200, B = 377.70

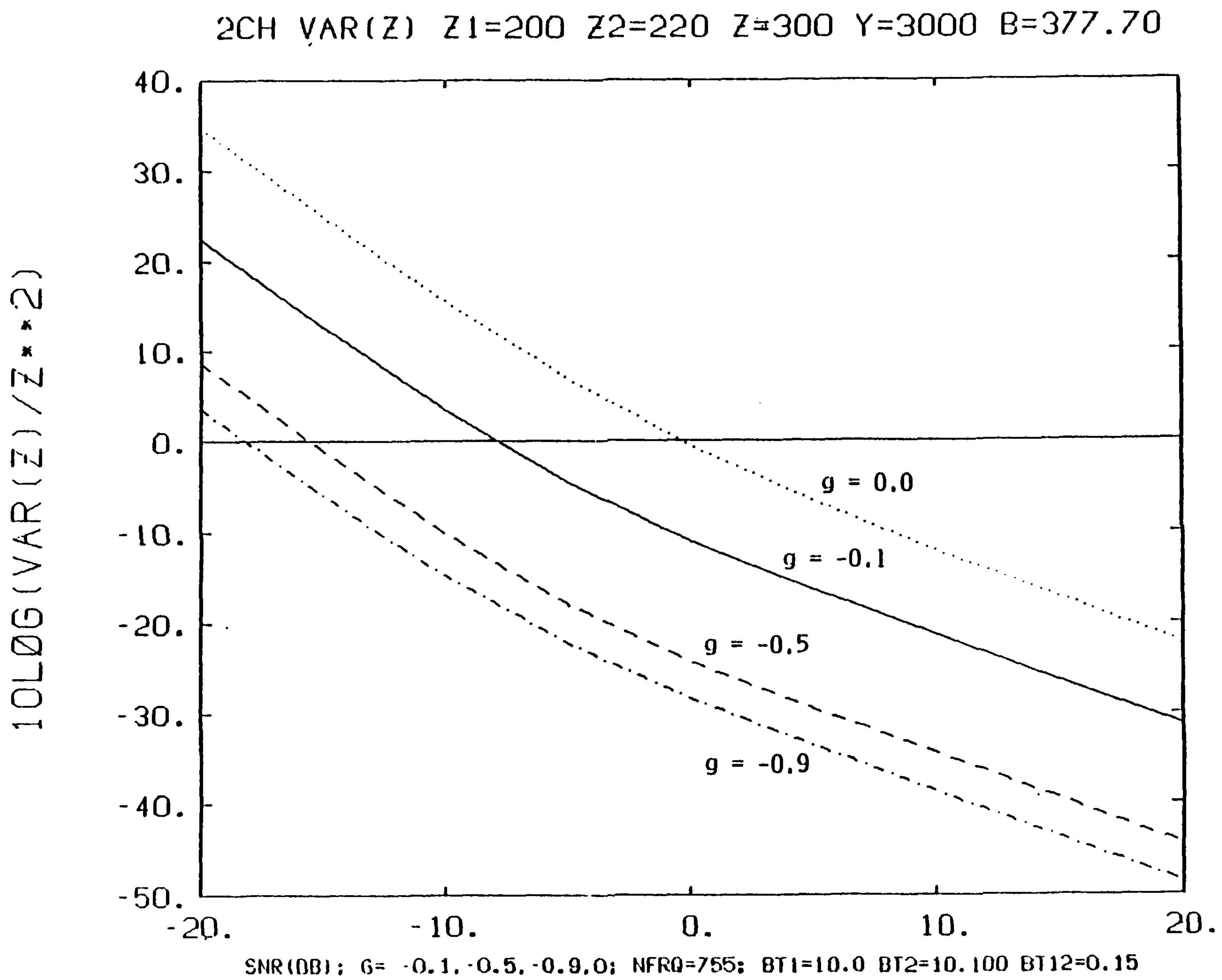


Figure 14: Two sensors with multipath -- z_2 = 220, B = 377.70

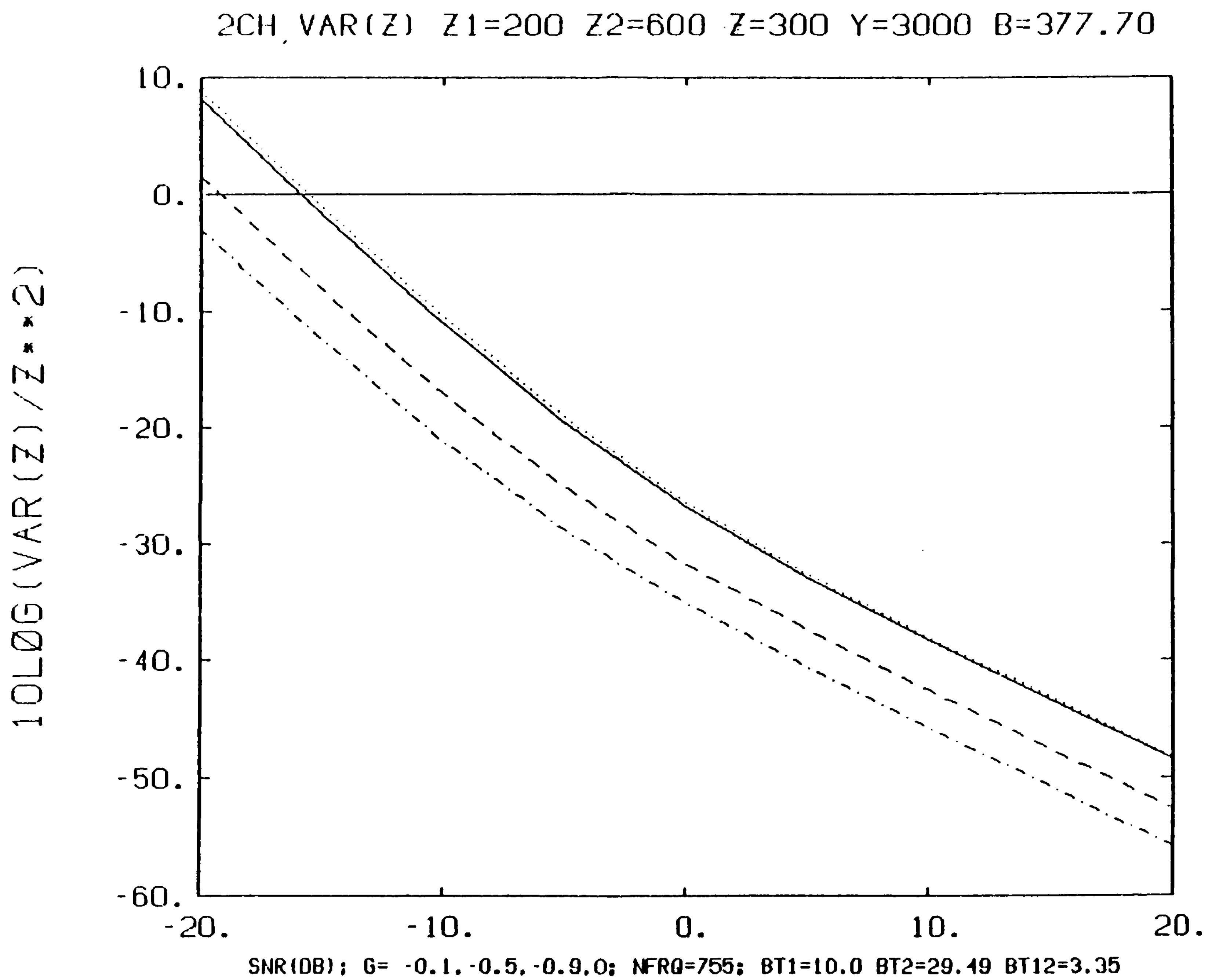


Figure 15: Two sensors with multipath -- z_2 = 600, B = 377.70

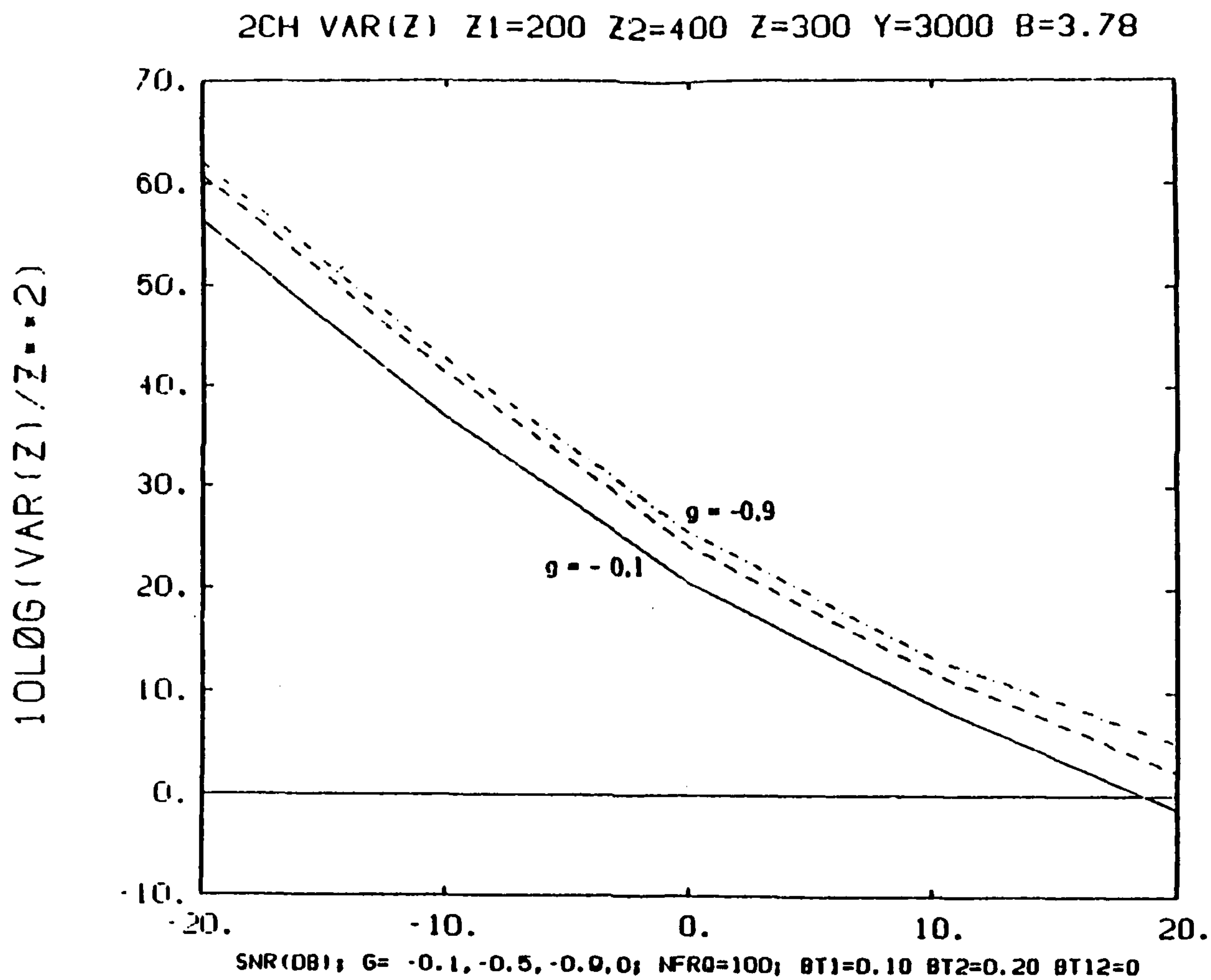


Figure 16: Unknown range -- z_2 = 400, B = 3.78

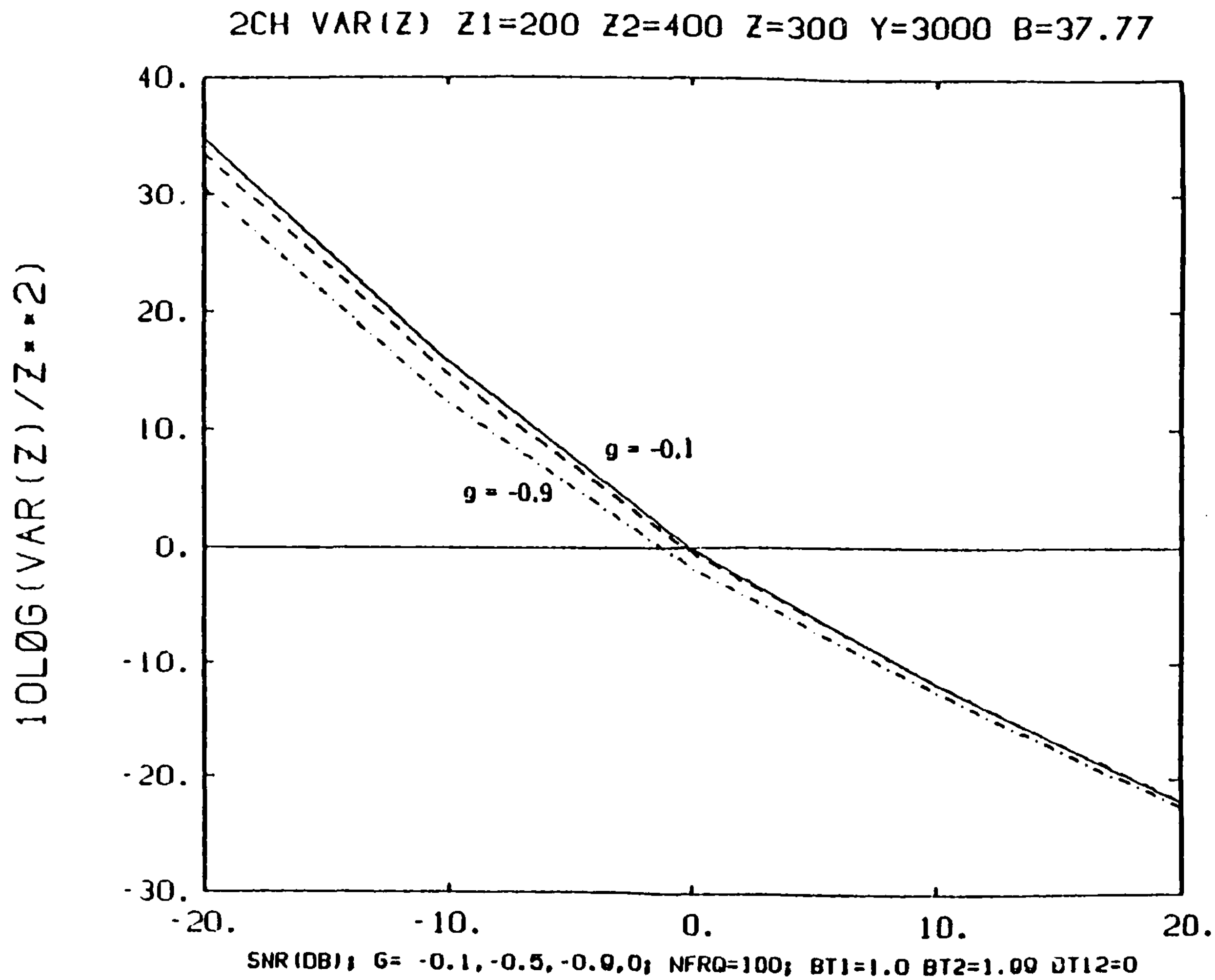


Figure 17: Unknown range -- z_2 = 400, B = 37.77

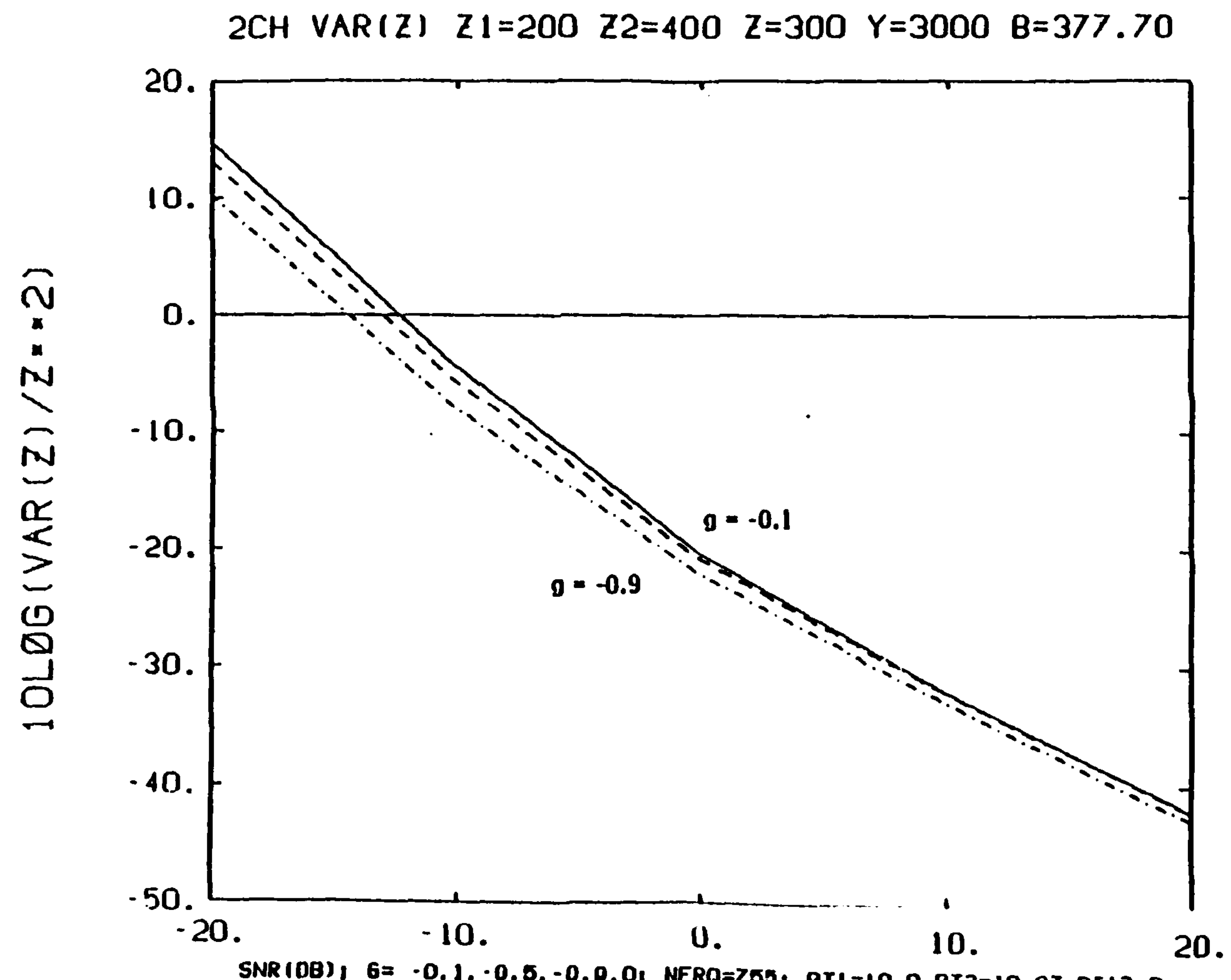


Figure 18: Unknown range -- z_2 = 400, B = 377.70

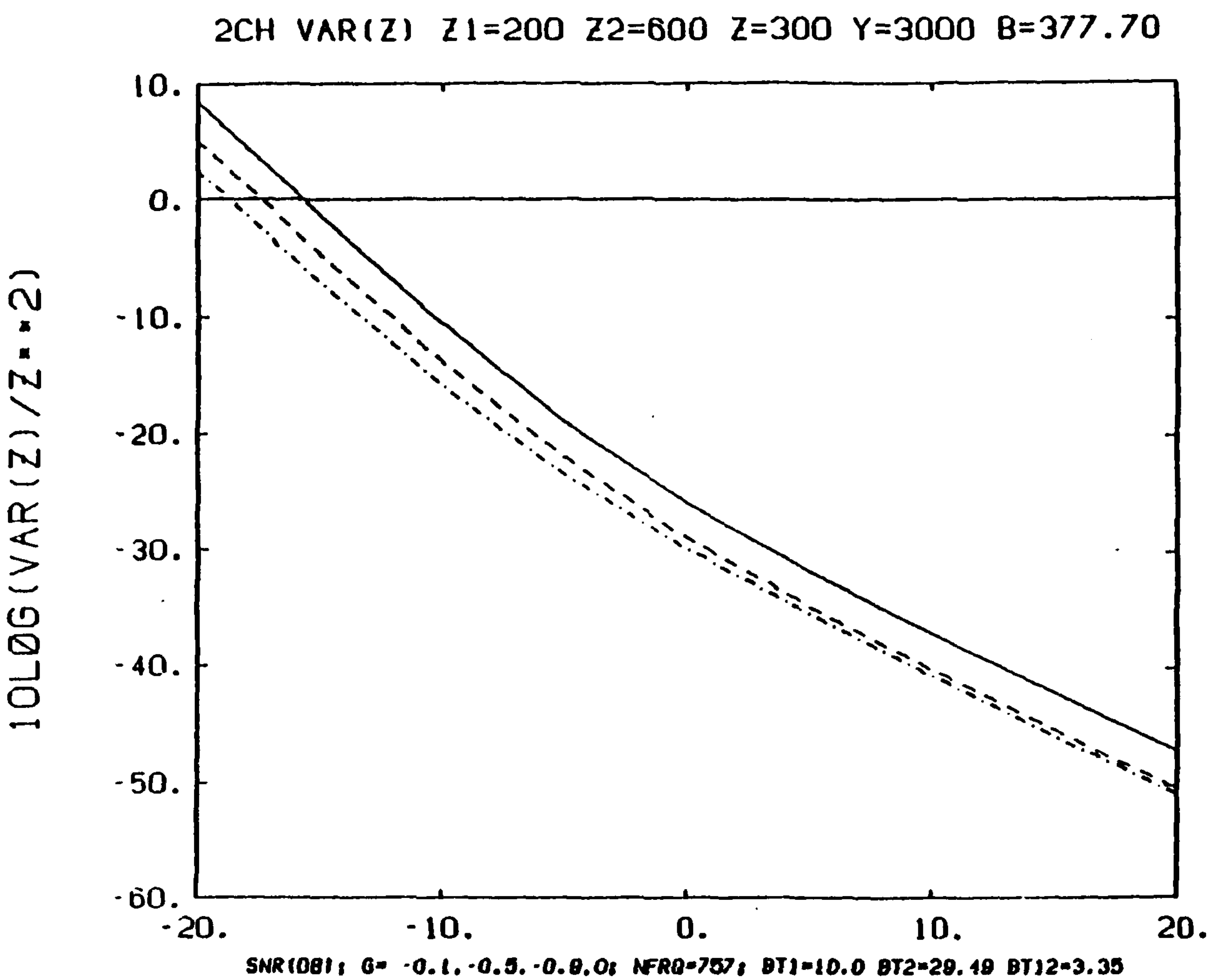


Figure 19: Unknown range -- z_2 = 600, B = 377.70

APPENDIX - SOFTWARE LISTINGS

The software developed to compute and display the CRB for the various test cases was written in the CtrlC language. CtrlC is a software product of Systems Control Technology. The CtrlC code below is contained on the SCT VAX directory NETVAX::dra2:[mts.bound.cr.depth]*.*. While connected to that directory, the monitor command @CMCM will run an example test case. The command @CMCM invokes the VAX/VMS DCL command file CMCM.COM which is also listed below. A summary of the various files is given first.

```
cmcm.com - Run CtrlC and feed it cmcm.ctr
cmcm.ctr - Outer loop of CtrlC commands for different SNR, BT, etc.
tdoa.ctr - Compute CRB for case of two-channel TDOA
scmp.ctr - Compute CRB for case of single-channel multipath data
mcmp.ctr - Compute CRB for case of multichannel multipath data
mctd.ctr - Compute multipath delays and tdoa from geometry
mpsn.ctr - Adjust SNR to eliminate signal boost due to multipath
simp.ctr - Simpson's rule for numerical integration
```

The file cmcm.com loads the CtrlC function library dra2:[MTS.ML.SPLIB]SP.LIB which contains many simple utility functions not implemented directly in CtrlC. The contents of SP.LIB are described in [Smith84b].

CMCM.COM - Initiation Of Test Cases

```
$! cmcm.com - Go into CtrlC, load SPLIB, and DO cmcm.CTR to obtain the
$! crb's for the 1-channel and 2-channel multipath situation
$!
$ run dra1:[sctlib.ctrlc]ctrlc.exe/nodebug
diary >cmcm.log;
lib DRA2:[MTS.ML.SPLIB]sp;
j = sqrt(-1);
// echo=1;
// term='vt12';
term='ptx';
hard = 'ptx';
redh >cmcm.ptx
do cmcm;
exit;
$ pra2 cmcm.ptx
```

```
$ pra2 ctrlc.ptx
$ pra2 cmcm.log
```

CMCM.CTR - Outer Loop Of CtrlC Commands For Different SNR, BT, Etc.

```
// []=cmcm()
deff mcmp
deff scmp
deff tdoa
deff mpsn
deff mctd
deff simp
dotdoa=0;
doscmp=0;
dompsnr=0;
z1=200;
snr=[-20,-10,0,10,20]; nsnr=5;
g=[-0.001,-0.01,-0.05]; ng=3;
vzpn=ones(nsnr,ng+1);
ntd=9; // max number of tdoa-only plots
vztd=ones(nsnr,ntd);
itd=1;
FOR z2=[600],..
  FOR z=[300],..
    FOR y=[3000],..
      FOR B=[10]/.026476,..
        [mpd1,mpd2,td12]=mctd(z1,z2,z,y);..
        bt1=mpd1*B;bt2=mpd2*B;bt3=abs(td12)*B;..
        str5=['; BT1=',cvfs(bt1,2),' BT2=',cvfs(bt2,2),' BT12=',cvfs(bt3,2)]
        Nfrq=max([2*round(b),100])+1;..
        str0=['; Nfrq=',cvs(round(Nfrq))];..
        disp([str5,str0]);..
        tmp1=[' z1=',cvs(z1)];..
        tmp2=[' z=',cvs(z),' y=',cvs(y),' b=',cvfs(b,2)];..
        IF dompsnr=1,tmp2=[tmp2,'C'];ELSE end;..
        str1=[tmp1,' z2=',cvs(z2),tmp2];..
        str4=[tmp1,tmp2];..
        gstr=' ';..
       IF DoTdoa=1, ..
          FOR i=1:nsnr,..
            vzpn(i,ng+1)=tdoa(snr(i),z1,z2,z,y,B);..
            IF itd<=ntd,vztd(i,itd)=vzpn(i,ng+1); ELSE end;..
            disp(['TDOA Var(z)=',cvs(.5*db(vzpn(i,ng+1))),' dB']);..
          end,..i
       ELSE end;..
        itd=itd+1;..
        FOR ig=1:ng,..
          tmp=cvfs(g(ig),1);..
```

```
            str2=[' g=',tmp];..
            gstr=[gstr,tmp];..
            IF ig<ng,gstr=[gstr,',']; ELSE end;..
            sstr=' ';..
            FOR i=1:nsnr,..
              tsnr=mpsn(snr(i),z1,z2,z,y,B,g(ig));..
              IF dompsnr=1,csnr=tsnr; ELSE csnr=snr(i); end;..
              tmp=[cvs(snr(i)),'(',cvfs(tsnr,2),')'];..
              str3=[' snr=',tmp];..
              sstr=[sstr,tmp];..
              IF i<nsnr,sstr=[sstr,',']; ELSE end;..
              disp([str1,str2,str3]);..
              IF doscmp=1,..
                vzsc(i,ig)=scmp(csnr,z1,z,y,B,g(ig),Nfrq);..
                disp(['1ch Var(z)=',cvs(0.5*db(vzsc(i,ig))),' dB']);..
              ELSE end;..
              [tmp,vypn,vyzpn]=mcmp(csnr,z1,z2,z,y,B,g(ig),Nfrq);..
              vzpn(i,ig)=tmp;..
              disp(['2ch Var(z)=',cvs(0.5*db(vzpn(i,ig))),' dB']);..
              disp(['2ch Var(y)=',cvs(0.5*db(vypn)),' dB']);..
              disp(['2ch Xv(zy)=',cvs(0.5*db(abs(vyzp))),' dB']);..
            end,..i
          end,..ig
          IF doscmp=1,..
            erase;..
            plot(snr,db(vzsc)/2);..
            title(['1ch Var(z)',str4]);..
            xlab(['SNR(dB); G=',gstr,str0,str5]);..
            ylab('10log(Var(z)/z**2)');..
            IF norm(term(1:3)-'ptx')<>0,replot; ELSE end;..
          ELSE end;..
          erase;..
          plot(snr,db(vzpn)/2);..
          title(['2ch Var(z)',str1]);..
          xlab(['SNR(dB); G=',gstr,',0',str0,str5]);..
          gstr=' ';..
          ylab('10log(Var(z)/z**2)');..
          IF norm(term(1:3)-'ptx')<>0,replot; ELSE end;..
        end,..B
      end,..y
    end,..z
  end, // z2
  IF DoTdoa=1, ..
    erase;..
    plot(snr,db(vztd)/2);..
    title(['All TDOA Var(z); LAST ',str4]);..
    xlab(['SNR(dB); LAST',str0,str5]);..
    ylab('10log(Var(z)/z**2)');..
    IF norm(term(1:3)-'ptx')<>0,replot; ELSE end;..
  ELSE end;
```

TDOA.CTR - Compute CRB For Case Of Two-channel TDOA

```
// [vzpn] = tdoa(snr, z1, z2, z, y, B)
//
//  Compute CRB for source depth given 2-channel TDOA measurements
//
// z1 = Sensor 1 depth
// z2 = Sensor 2 depth
// z = Target depth
// y = Target range projected to surface (meters)
// B = Target bandwidth (Hz)
// Snr = Signal to noise ratio in dB at each sensor

 c = 1500; // Speed of sound (m/sec)
 snr1 = 10**(snr/10); snr2 = 10**(snr/10);
 t11 = sqrt(y*y+(z-z1)**2)/c;
 t12 = sqrt(y*y+(z-z2)**2)/c;
 d = t12 - t11;
 dztmp = z1*t12-z2*t11-z*d;
 if abs(dztmp)<eps, ..
   disp(' tdoa: numerical failure. tdoa not a function of depth'); ..
   vzpn=1.0; .. create line at 0dB ..
   return;
 dzdd = c*c*t11*t12/dztmp;
 vard = (3/(2*pi*B)**2)*(1+snr1+snr2)/(snr1*snr2);
 vzpn = (vard*dzdd**2)/(z*z);
```

SCMP.CTR - Compute CRB For Case Of Single-channel Multipath Data

```
// [vzpn] = scmp(snr, z1, z, y, B, g, Nfrq)
//
//  Compute per-sample CRB for 1-channel multipath situation
//  Sampling rate is assumed to be 2B rad/sec
//
// z1 = Sensor 1 depth
// z = Target depth
// y = Target range projected to surface (meters)
// B = Target bandwidth (Hz)
// g = Multipath attenuation
// Snr = Signal to noise ratio in dB
// Nfrq = Number of frequency samples uniform on [0,B]
//
// Derived constants

 c = 1500; cs = c*c;                      // Speed of sound (m/sec)
 f = b*[0:Nfrq-1]/Nfrq;                          // Hertz frequency axis
 w = 2*pi*f;                              // Radian frequency axis
 Sx=1; Se = 10**(-snr/10);                // PSD in band [-B,B]
```

```
t11 = sqrt(y*y+(z-z1)**2)/c;
t21 = sqrt(y*y+(z+z1)**2)/c;
D = t21-t11;
dDdz = (z+z1)/(cs*t21) - (z-z1)/(cs*t11);
Sy = ((1+g**2)*ONES(1,Nfrq)+2*g*cos(w*D))*Sx + Se*ones(1,Nfrq);
SydD = -2*Sx*g*w.*sin(w*D);
si = SydD./(Sy+EPS*ONES(1,Nfrq));// Specific information for delay estimat
JD = SUM(si.*si)/(2*Nfrq);      // per-sample CRB Var(D) .ge. 1/JD
vzpn = (1/(JD*(dDdz**2)+EPS))/(z*z);
```

MCMP.V2 - CRB For Multichannel Multipath Data, Known Range

```
// [vzpn] = mcmp(snr,z1,z2,z,y,B,g,Nfrq)
//
//  Compute CRB for 2-channel multipath situation
//
// z1 = Sensor 1 depth
// z2 = Sensor 2 depth
// z = Target depth
// y = Target range projected to surface (meters)
// B = Target bandwidth (Hz). Must be from 0 to .5
// g = Common multipath attenuation
// Snr = Signal to noise ratio in dB
// Nfrq = Number of frequency samples uniform on [0,B]
//
// Derived constants

 j = sqrt(-1);
 g1 = g;         // Relative attenuation of 2ndary path to channel 1
 g2 = g;         // Relative attenuation of 2ndary path to channel 2
 c = 1500;       // Speed of sound (m/sec)
 f = b*[0:Nfrq-1]/Nfrq;                          // Hertz frequency axis
 w = 2*pi*f;                            // Radian frequency axis
 Sx1=1; Sx2=1;                          // PSD of signal in band [-B,B]
 snr1 = snr; snr2 = snr;                // SNR in channels 1 and 2, resp.
 Se1 = 10**(-snr1/10);                  // PSD of noise in band [-pi,pi]
 Se2 = 10**(-snr2/10);                  // PSD of noise in band [-pi,pi]
// disp('disabling channel 1'); Sx1=0;

// Multipath time delays

 t11 = sqrt(y*y+(z-z1)**2)/c;
 t21 = sqrt(y*y+(z+z1)**2)/c;
 t12 = sqrt(y*y+(z-z2)**2)/c;
 t22 = sqrt(y*y+(z+z2)**2)/c;
 BT = max([(t22-t12)*B,(t21-t11)*B]);
 ppc = Nfrq/BT; // Number of integration points per multipath spectral cycle
 if ppc<20, disp([' Warning, BT=',cvfs(BT,2),' while Nfrq=',cvs(Nfrq)]); ..
   disp([' which means only ',cvfs(ppc,2),' integration points per cycle']);
```

```
// derivatives of delay wrt z

 cs = c*c;
 t11z = (z-z1)/(cs*t11);
 t21z = (z+z1)/(cs*t21);
 t12z = (z-z2)/(cs*t12);
 t22z = (z+z2)/(cs*t22);

// Derivatives of source-to-sensor spectra wrt delays

 // s1/z = s1/t11 t11/z + s1/t21 t21/z
 s111 = -(2*Sx1*g1)*(w.*sin((t11-t21)*w));      // s1/t11 = - s1/t21
 s1z = s111*(t11z-t21z);                         // s111*t11z + s121*t21z

 // s2/z = s2/t12 t12/z + s2/t22 t22/z
 s212 = -(2*Sx2*g2)*(w.*sin(w*(t12-t22)));      // = -s222
 s2z = s212*(t12z - t22z);

 // s12z = conj(s21z) = sum(i,j=1,2) s12/tij tij/z
 ejw = exp(j*w);
 ej11 = exp(j*w*t11);
 ej12 = exp(j*w*t12);
 ej21 = exp(j*w*t21);
 ej22 = exp(j*w*t22);
 k1=ej12+g2*ej22;
 k2=conj(ej11+g1*ej21);
 t=j*w*sqrt(Sx1*Sx2);
 x11 = -t.*conj(ej11).*k1;      // x expands to "S12/t"
 x21 = -g1*t.*conj(ej21).*k1;
 x12 = t.*ej12.*k2;
 x22 = g2*t.*ej22.*k2;
 s12z = x11*t11z + x12*t12z + x21*t21z + x22*t22z;

 s1 = Sx1*abs(ones(1,Nfrq)+g1*ej11.*conj(ej21))**2 + Se1*ones(1,Nfrq);
 s2 = Sx2*abs(ones(1,Nfrq)+g2*ej12.*conj(ej22))**2 + Se2*ones(1,Nfrq);
 s12 = sqrt(Sx1*Sx2)*k2.*k1;

// Do sisz = S**(-1) S/z

// dl means d ln ... dlij means S^-1 dS/dz [i,j]
 detS = s1.*s2 - s12.*conj(s12);
 dl11 = (s2.*s1z-s12.*conj(s12z))./detS;
 dl12 = (s2.*s12z-s12.*s2z)./detS;
 dl21 = (s1.*conj(s12z)-conj(s12).*s1z)./detS;
 dl22 = (s1.*s2z-conj(s12).*s12z)./detS;

 // Trace dl**2

 tdls = dl11.*dl11 + dl22.*dl22 + 2*dl12.*dl21;
 stdl = max([eps,abs(sum(tdls))]);
 if stdl=eps, disp('numerical failure. vanishing info matrix');
 sl = sum(tdls(1:2:Nfrq));
```

```
s2 = sum(tdls(2:2:Nfrq));
errv = [(s1-s2)/stdl, abs(s1+s2)-stdl];
if sum(abs(errv))>0.01, ..
  disp('[relative half-sum difference, 0] (numerical integration check):')
  errv, ..
else end;
tinf = stdl/(2*Nfrq);
vzpn = (1/tinf)/(z*z);
```

MCMP.CTR - CRB For Multichannel Multipath Data, Unknown Range

```
// [vzpn,vypn,vyzp] = mcmp(snr,z1,z2,z,y,B,g,Nfrq)
//
//  Compute CRB for range and depth estimation given 2-channel
//     multipath and tdoa measurements.
//
// z1 = Sensor 1 depth
// z2 = Sensor 2 depth
// z = Target depth
// y = Target range projected to surface (meters)
// B = Target bandwidth (Hz). Must be from 0 to .5
// g = Common multipath attenuation
// Snr = Signal to noise ratio in dB
// Nfrq = Number of frequency samples uniform on [0,B]
//
// vzpn = Cramer-Rao lower bound for the estimation of z
// vypn = Cramer-Rao lower bound for the estimation of y
// vyzp = Cramer-Rao lower bound for cross-variance of y and z
//
// Derived constants

 j = sqrt(-1);
 g1 = g;        // Relative attenuation of 2ndary path to channel 1
 g2 = g;        // Relative attenuation of 2ndary path to channel 2
 c = 1500;      // Speed of sound (m/sec)
 f = b*[0:Nfrq-1]/Nfrq;                          // Hertz frequency axis
 w = 2*pi*f;                            // Radian frequency axis
 Sx1=1; Sx2=1;                          // PSD of signal in band [-B,B]
 snr1 = snr; snr2 = snr;                // SNR in channels 1 and 2, resp.
 Se1 = 10**(-snr1/10);                  // PSD of noise in band [-pi,pi]
 Se2 = 10**(-snr2/10);                  // PSD of noise in band [-pi,pi]
// disp('disabling channel 1'); Sx1=0;

// Multipath time delays

 t11 = sqrt(y*y+(z-z1)**2)/c;
 t21 = sqrt(y*y+(z+z1)**2)/c;
 t12 = sqrt(y*y+(z-z2)**2)/c;
 t22 = sqrt(y*y+(z+z2)**2)/c;
```

```
BT = max([(t22-t12)*B,(t21-t11)*B]);
ppc = Nfrq/BT; // Number of integration points per multipath spectral cycl
if ppc<20, disp([' Warning, BT=',cvfs(BT,2),' while Nfrq=',cvs(Nfrq)]); ..
  disp([' which means only ',cvfs(ppc,2),' integration points per cycle']);

// derivatives of delay wrt z

cs = c*c;
t11z = (z-z1)/(cs*t11);
t21z = (z+z1)/(cs*t21);
t12z = (z-z2)/(cs*t12);
t22z = (z+z2)/(cs*t22);

// derivatives of delay wrt y

cs = c*c;
t11y = y/(cs*t11);
t21y = y/(cs*t21);
t12y = y/(cs*t12);
t22y = y/(cs*t22);

// Derivatives of source-to-sensor spectra wrt delay, depth, and range

// s1/z = s1/t11 t11/z + s1/t21 t21/z
s111 = -(2*Sx1*g1)*(w.*sin((t11-t21)*w));      // s1/t11 = - s1/t21
s1z = s111*(t11z-t21z);                        // s111*t11z + s121*t21z
s1y = s111*(t11y-t21y);                        // s111*t11y + s121*t21y

// s2/z = s2/t12 t12/z + s2/t22 t22/z
s212 = -(2*Sx2*g2)*(w.*sin(w*(t12-t22)));      // = -s222
s2z = s212*(t12z - t22z);
s2y = s212*(t12y - t22y);

// s12z = conj(s21z) = sum(i,j=1,2) s12/tij tij/z
ejw = exp(j*w);
ej11 = exp(j*w*t11);
ej12 = exp(j*w*t12);
ej21 = exp(j*w*t21);
ej22 = exp(j*w*t22);
k1=ej12+g2*ej22;
k2=conj(ej11+g1*ej21);
t=j*w*sqrt(Sx1*Sx2);
x11 = -t.*conj(ej11).*k1;       // x expands to "S12/t"
x21 = -g1*t.*conj(ej21).*k1;
x12 = t.*ej12.*k2;
x22 = g2*t.*ej22.*k2;
s12z = x11*t11z + x12*t12z + x21*t21z + x22*t22z;
s12y = x11*t11y + x12*t12y + x21*t21y + x22*t22y;

s1 = Sx1*abs(ones(1,Nfrq)+g1*ej11.*conj(ej21))**2 + Se1*ones(1,Nfrq);
s2 = Sx2*abs(ones(1,Nfrq)+g2*ej12.*conj(ej22))**2 + Se2*ones(1,Nfrq);
s12 = sqrt(Sx1*Sx2)*k2.*k1;
```

```
detS = s1.*s2 - s12.*conj(s12);

// Do sisz = S**(-1) S/z. Below, dzij means S^-1 dS/dz [i,j]
 dz11 = (s2.*s1z-s12.*conj(s12z))./detS;
 dz12 = (s2.*s12z-s12.*s2z)./detS;
 dz21 = (s1.*conj(s12z)-conj(s12).*s1z)./detS;
 dz22 = (s1.*s2z-conj(s12).*s12z)./detS;

// Trace dz**2
 tdzs = dz11.*dz11 + dz22.*dz22 + 2*dz12.*dz21;
 stdz = simp(tdzs);
 ctdz = simp(tdzs(1:2:Nfrq));
 disp(' J11: [simp(n)-simp((n-1)/2)]/simp(n)] = ');
 (stdz-ctdz)/max([stdz,eps])
 disp(' J11: [simp(n)-simp((n-1)/2)] = ');
 stdz-ctdz
 if abs(imag(stdz))>eps, disp('reality failure, depth part'); stdz
 stdz = real(stdz);
 Jmtx = ONES(2); // Allocate Fisher information matrix
 Jmtx(1,1) = stdz/2;

// Do sisy = S**(-1) S/y. Below, dyij means S^-1 dS/dy [i,j]
 dy11 = (s2.*s1y-s12.*conj(s12y))./detS;
 dy12 = (s2.*s12y-s12.*s2y)./detS;
 dy21 = (s1.*conj(s12y)-conj(s12).*s1y)./detS;
 dy22 = (s1.*s2y-conj(s12).*s12y)./detS;

// Trace dy**2
 tdys = dy11.*dy11 + dy22.*dy22 + 2*dy12.*dy21;
 stdy = simp(tdys);
 ctdy = simp(tdys(1:2:Nfrq));
 disp(' J22: [simp(n)-simp((n-1)/2)]/simp(n)] = ');
 (stdy-ctdy)/max([stdy,eps])
 disp(' J22: [simp(n)-simp((n-1)/2)] = ');
 stdy-ctdy
 if abs(imag(stdy))>eps, disp('reality failure, range part'); stdy
 stdy = real(stdy);
 Jmtx(2,2) = stdy/2;

// Trace dz*dy
 tzys = dz11.*dy11 + dz12.*dy21 + dz21.*dy12 + dz22.*dy22;
 stzy = simp(tzys);
 ctzy = simp(tzys(1:2:Nfrq));
 disp(' J12: [simp(n)-simp((n-1)/2)]/simp(n)] = ');
 (stzy-ctzy)/max([abs(stzy),eps])
 disp(' J12: [simp(n)-simp((n-1)/2)] = ');
 stzy-ctzy
 if abs(imag(stzy))>eps, disp('reality failure, range part');
 stzy = real(stzy);
 if abs(stzy)<eps, disp('mcmp: zero off-diagonal in info matrix');
 Jmtx(1,2) = stzy/2;
 Jmtx(2,1) = Jmtx(1,2);
```

```
long; Jmtx
short;
disp('condition number of J matrix:'); COND(Jmtx)

Jinv = INV(Jmtx);
vzpn = Jinv(1,1)/(z*z);
vypn = Jinv(2,2)/(y*y);
vyzp = Jinv(1,2)/(y*z);
```

MCTD.CTR - Compute Multipath Delays And Tdoa From Geometry

```
// [mpd1,mpd2,td12] = mctd(z1,z2,z,y)
//
//  Given geometry, compute multipath delays in each channel,
//  and tdoa between channels.
//
// z1 = Sensor 1 depth
// z = Target depth
// y = Target range projected to surface (meters)
//
 c = 1500; cs = c*c;                     // Speed of sound (m/sec)
 t11 = sqrt(y*y+(z-z1)**2)/c;            // source to sensor 1 direct
 t21 = sqrt(y*y+(z+z1)**2)/c;            // source to sensor 1 reflected
 mpD1 = t21-t11;                         // multipath delay, sensor 1
 t12 = sqrt(y*y+(z-z2)**2)/c;            // source to sensor 2 direct
 t22 = sqrt(y*y+(z+z2)**2)/c;            // source to sensor 2 reflected
 mpD2 = t22-t12;                         // multipath delay, sensor 2
 td12 = t12-t11;                         // tdoa, channel 1 minus 2
```

MPSN.CTR - Adjust SNR To Eliminate Signal Boost Due To Multipath

```
// [csnr] = mpsn(snr,z1,z2,z,y,B,g)
//
//  Compute SNR correction due to signal power gain
//  in the presence of multipath.
//
// z1 = Sensor 1 depth
// z = Target depth
// y = Target range projected to surface (meters)
// B = Target bandwidth (Hz)
// g = Multipath attenuation
// Snr = Signal to noise ratio in dB given no multipath
//
// Derived constants
```

```
c = 1500; cs = c*c;                          // Speed of sound (m/sec)
W = 2*pi*B;                                  // Radian band limit

t11 = sqrt(y*y+(z-z1)**2)/c;                 // source to sensor 1 direct
t21 = sqrt(y*y+(z+z1)**2)/c;                 // source to sensor 1 reflected
D1 = t21-t11;                                // multipath delay, sensor 1
WD1 = W*D1;
fac1 = 1+g**2+2*g*SIN(WD1)/WD1;

t12 = sqrt(y*y+(z-z2)**2)/c;                 // source to sensor 2 direct
t22 = sqrt(y*y+(z+z2)**2)/c;                 // source to sensor 2 reflected
D2 = t22-t12;                                // multipath delay, sensor 2
WD2 = W*D2;
fac2 = 1+g**2+2*g*SIN(WD2)/WD2;

if snr=-10, if abs(g)=0.1, disp('BT for MP1,MP2, and TDOA:'), ..
 [B*D1,B*D2,B*abs(t12-t11)]

fac = sqrt(fac1*fac2);                                  // I don't know if this is b
if fac<eps, fac=eps; disp('mpsn: numerical failure: zero delay');

csnr = snr/fac;                              // corrected SNR
```

SIMP.CTR - Simpson's Rule For Numerical Integration

```
// [int] = simp(f)
// integrate f using Simpson's rule. Integral is NORMALIZED.
n = max(size(f));
if round(n/2)*2=n, disp('simp: Need odd number of points');
if n<3, disp('simp: Need at least 5 data points');
int = (f(1)+f(n)+4*sum(f(2:2:n-1))+2*sum(f(3:2:n-2)))/3;
int = int/(n-1);
```

APPENDIX 4

CORRELOGRAM GENERATION PROGRAM

The following are listings of the correlogram and line tracking software.

```
        options/check=all

        program xrsda
c
c       XRSDA takes .i2 data from xfile and yfile and produces auto-
c       and cross- correlograms and spectrograms stored as .r4 in
c       rxyfile, sxyfile, rxxfile, etc.
c
c       The ith line of the cross-correlogram is +/-1024 lags of the
c       biased/unbiased ML/PHAT/SCOT/unormalized cross-correlation estimate
c       based on nspl*nrps*256 points, beginning at the (d*i)th point
c       of x and y.  The lines of the auto-correlograms are 1024 lags
c       of the auto-correlation estimate.
c
c       Correlograms are computed as ifft(spectrograms).  The spectrograms
c       are computed as peroidograms based on nspl non-overlapping averages
c       of nrps*256 point 8k fft's of x and y.
c
c
c       XRSDA 'decimates' the correlograms and spectrograms for display on
c       the DeAnza.  r42mas, daform and flx should be used to create plot
c       files on floppy disks.
c
c
c
c
c       declarations
c
c
        implicit none                           !! input/computation variables
        character*20 xfile,yfile,rxyfile,sxyfile
        character*20 rxxfile,sxxfile,ryyfile,syyfile
        character*16 nrmlzn
        integer i,ii,iii,j,k,l,itemp
        integer nspl,nrps,sdf,m,d,nl,nrpl,frr
        integer lpco,hpco,nrm
        integer*2 x(256,2048),y(256,2048)
        real*4 r(2048,3),s(512,3),rtemp(3),scl(3)
        logical gtyes,xey,bce
c
c
c
        real*4 temp(1024*64)            !! ****
        integer a,b,c                   !! ****
c
c
c
c
        integer status,ierr                     !! ap variables
        integer xadr,xxadr,yadr,yyadr,wadr,sadr
        integer xyadr,tempadr,fts,nc,ne
        real*4 apbufr(8192,3),apbufs(4097,3),apbufw(8192)
c
c
        integer xch,ych,sxych,rxych             !! i/o variables
```

```
        integer sxxch,rxxch,syych,ryych
c
c
c       initialization
c
c
        xfile='bvdata:mpc1.i2'                  !! set default input values
        yfile='bvdata:mpc2.i2'
        rxyfile='cg:rxy.r4'
        sxyfile='cg:sxy.r4'
        rxxfile='cg:rxx.r4'
        sxxfile='cg:sxx.r4'
        ryyfile='cg:ryy.r4'
        syyfile='cg:syy.r4'
        xey=.false.
        nrmlzn='nonePHATSCOTML  '
c
        nl=512
        nrps=16
        m=512
        sdf=8
        d=1
        nspl=4
        lpco=4096
        hpco=1
        frr=1
        nrm=0
c
        xadr=1000               !! ap defaults
        xxadr=9200
        yadr=17400
        yyadr=25600
        xyadr=33800
        tempadr=42000
        sadr=50200
        wadr=55000
c
        nc=4097
        fts=8192
c
        a=0                     !! ****
        b=1024*64-1             !! ****
        c=2                     !! ****
c
        xch=1                   !! i/o defaults
        ych=2
        rxych=3
        sxych=4
        rxxch=13
        sxxch=14
        ryych=23
        syych=24
c
c
c       input
c
c
200     call clrscrn
```

```
        call otstr('Program XRSDA -- computes the atuo- and cross-',1)
        call otstr('                 correlogram and spectrogram',1)
        call otstr('                 of x,y. Output for DeAnza',5)

        call gtstr('Enter the name of the input file for x ',xfile,20)
        call gtstr('Enter the name of the input file for y ',yfile,20)
        call gtstr('Enter the output file-name for Rxy',rxyfile,20)
        call gtstr('Enter the output file-name for Sxy',sxyfile,20)
        call gtstr('Enter the output file name for Rxx',rxxfile,20)
        call gtstr('Enter the output file name for Sxx',sxxfile,20)
        call gtstr('Enter the output file name for Ryy',ryyfile,20)
        call gtstr('Enter the output file name for Syy',syyfile,20)
        call otstr(' ',2)
        call gtint('Number of records per processing segment',nrps,16,2)
        itemp=int(2048./float(nrps))
        call gtint('Number of segments per correlogram line',nspl,itemp,1)
        call gtint('Number of records skipped between lines',d,99999,1)
        call gtint('Number of the first record read',frr,99999,1)
        call gtint('Number of lines computed',nl,512,1)
        call otstr(' ',2)
        call gtint('0-no normalization, 1-PHAT, 2-SCOT, 3-ML',nrm,3,0)
        bce=gtyes('Biased correlation estimates (y/n):')
        call otstr(' ',2)
        call gtint('Enter low pass cut off -- 4096 bins ',lpco,4097,1)
        call gtint('Enter high pass cut off -- 4096 bins',hpco,lpco,0)
c

        call clrscrn                              !! print input values
        call otstr('Data entered:',2)
        write(6,120) xfile
 120    format(' data source 1:',a20)
        write(6,130) yfile
 130    format(' data source 2:',a20)
        write(6,140) rxyfile
 140    format(' Rxy:',a20)
        write(6,150) sxyfile
 150    format(' Sxy:',a20)
        write(6,141) rxxfile
 141    format(' Rxx:',a20)
        write(6,142) sxxfile
 142    format(' Sxx:',a20)
        write(6,143) ryyfile
 143    format(' Ryy:',a20)
        write(6,144) syyfile
 144    format(' Syy:',a20)
        call otstr('S',1)
        write(6,160) (nspl*nrps)
 160    format(' # of 256 pt records per correlogram line :',i6)
        write(6,170) d
 170    format(' # of 256 pt records skipped between lines:',i6)
        write(6,196) (frr-1)
 196    format(' # of records skipped before the 1st line :',i6)
        write(6,180) nl
 180    format(' # of correllogram lines           :',i4)
        write(6,194) nrps
 194    format(' # of records processed per 8k fft:',i4)
        call otstr('#',1)
        write(6,192) (nrmlzn(nrm*4+1:nrm*4+4))
```

```
192     format(' power specturm normalization: ',a4)
        write(6,193) bce
193     format(' biased correllation estimates:',l2)
        call otstr('b',1)
        write(6,195) hpco/2,lpco/2
195     format(' correlogram passband -- (',i5,',',i5,') Hz')
        call otstr('c',2)
c
c
        if (gtyes('Values OK (y/n):') .eq. .false.) go to 200
c
c                                                   !! reenter if mistakes
c
c
c
c
c       post-input initalization
c
c
        if (xfile .eq. yfile) xey=.true.              !! auto/cross flag
c
        ne=nrps*256
        nrpl=nrps*nspl
c
c
        do i=1,8192
        apbufw(i)=0.
        enddo
c
        do i=1,ne-1
        apbufw(i)=1.0
        apbufw(8193-i)=1.0
        if (.not.bce) then
                apbufw(i)=float(ne)/float(ne-i+1)
                apbufw(8193-i)=float(ne)/float(ne-i)
        endif
        enddo
c
c
c
c
c       open files
c
c
        call opn(rxych,rxyfile,'new')                 !! output files
        call opn(sxych,sxyfile,'new')
        call opn(rxxch,rxxfile,'new')
        call opn(sxxch,sxxfile,'new')
        call opn(ryych,ryyfile,'new')
        call opn(syych,syyfile,'new')
c
c
        call opn(xch,xfile,'old')                     !! open xfile
        if (.not. xey) call opn(ych,yfile,'old')      !! open y--if necessary
c
c
c
c
```

```
c       main loop -- get data, compute correlogram lines
c
c
        do 230 j=1,nl
c
c
c
        if (j .eq. 1) then                !! first data set -- read
c                                         !! nrpl records
c
                call rrecxy(xch,x(1,1),ych,y(1,1),frr,frr+nrpl-1,xey)
c
        else                              !! not first data set --
                if (d .ge. nrpl) then     !! d># -- fresh data
c
                        ii=(j-1)*d+frr
                        iii=ii+nrpl-1
                        call rrecxy(xch,x(1,1),ych,y(1,1),ii,iii,xey)
c
                else                      !! make room for d
                                          !! records new data
                        do 260 i=1,nrpl-d
                        do 270 ii=1,256
                        y(ii,i)=y(ii,i+d)
270                     x(ii,i)=x(ii,i+d)
260                     continue
                                          !! get d new records
                                          !! put at end of x,y
                        iii=(j-1)*d+nrpl+frr
                        ii=iii-d+1
                        i=nrpl-d+1
                        call rrecxy(xch,x(1,i),ych,y(1,i),ii,iii,xey)
                endif
        endif
c
c
c       ap computations
c
c
        if (j.eq.1) then
                call apinit(0,0,status)           !! initalize ap
                call vclr(0,1,65535)              !! clear ap
                call apwr                         !! load constants
                call apput(apbufw,wadr,8192,2)
                call apwd
        else
                call vclr(1000,1,53500)           !! clear all but window
        endif
c
c
        do l=1,nspl                       !! process x,y

        call vclr(xadr,1,2*ne)            !! clear data area
        call vclr(yadr,1,2*ne)
```

```
c
c
      call apwr                              !! x,y --> ap
      call apput(x(1,(l-1)*nrps+1),xadr,ne,1)
      call apput(y(1,(l-1)*nrps+1),yadr,ne,1)
      call apwd
c
c
      call vflt(xadr,1,xadr,1,ne)            !! convert to reals
      call vflt(yadr,1,yadr,1,ne)
c
c
      call rfft(xadr,fts,1)                  !! take 8k fft's
      call rfft(yadr,fts,1)
      call rfftsc(xadr,fts,3,1)              !! scale/undo wierd packing
      call rfftsc(yadr,fts,3,1)
c
c
      call vmov(xyadr,1,tempadr,1,fts+2)     !! conj(X)*Y+sum --> xyadr
      call cvma(xadr,2,yadr,2,tempadr,2,xyadr,2,nc,-1)
c
c
      call vmov(xxadr,1,tempadr,1,nc)        !! conj(X)*X+sum --> xxadr
      call scjma(xadr,2,tempadr,1,xxadr,1,nc)
c
c
      call vmov(yyadr,1,tempadr,1,nc)        !! conj(Y)*Y+sum --> yyadr
      call scjma(yadr,2,tempadr,1,yyadr,1,nc)
c
c
c
      enddo                                          !! get new x,y
c
c
c
      if (nrm.eq.2) then                                     !! SCOT
c
              call vmul(yyadr,1,xxadr,1,tempadr,1,nc)
              call vsqrt(tempadr,1,sadr,1,nc)        !! (xx*yy)**0.5
              call crvdiv(xyadr,2,sadr,1,tempadr,2,nc)       !! SCOT
              call vmov(tempadr,1,xyadr,1,2*nc)
c
      endif
c
c
c
      if (nrm.eq.1) then                                     !! PHAT
c
              call vmov(xyadr,1,tempadr,1,nc*2)
              call cvmags(tempadr,2,xyadr,1,nc)      !! magnitude**2
              call vsqrt(xyadr,1,sadr,1,nc)          !! sqrt(mag**2)
              call crvdiv(tempadr,2,sadr,1,xyadr,2,nc)       !! PHAT
c
      endif
c
c
c
      if (nrm.eq.3) then                             !! ML
```

```
c
      call cvmags(xyadr,2,sadr,1,nc)                    !! mag**2(xy)
      call vsqrt(sadr,1,tempadr,1,nc)                   !! mag(xy)
      call vclr(xadr,1,2*nc)
      call crvdiv(xyadr,2,tempadr,1,xadr,2,nc)          !! phase(xy)
      call vclr(yadr,1,2*nc)
      call vmul(xxadr,1,yyadr,1,yadr,1,nc)              !! xx*yy
      call vsub(sadr,1,yadr,1,xyadr,1,nc)               !! xx*yy-xy**2
      call vdiv(xyadr,1,sadr,1,tempadr,1,nc)            !! xy**2/(...)
      call crvmul(xadr,2,tempadr,1,xyadr,2,nc)          !! ML
c
      endif
c
c
      call vmov(xyadr,1,tempadr,1,fts+2)                !! mag of Sxy
      call cvmags(tempadr,2,sadr,1,nc)
      call vmov(sadr,1,tempadr,1,nc)
      call vsqrt(tempadr,1,sadr,1,nc)
c
c
      call apwr
      call apget(apbufs(1,1),sadr,nc,2)                 !! xy-->apbuf
      call apget(apbufs(1,2),xxadr,nc,2)                !! xx-->apbuf
      call apget(apbufs(1,3),yyadr,nc,2)                !! yy-->apbuf
      call apwd
c
c
c
c
      if (hpco.ne.0) then                               !! lp filter
              call vclr(xyadr,1,2*hpco)                 !! Sxy, Sxx, Syy
              call vclr(xxadr,1,hpco)
              call vclr(yyadr,1,hpco)
      endif
c
c
      if (lpco.ne.4097) then                            !! hp filter
              call vclr(xyadr+2*lpco,1,2*nc-2*lpco)     !! Sxy, Sxx, Syy
              call vclr(xxadr+lpco,1,nc-lpco)
              call vclr(yyadr+lpco,1,nc-lpco)
      endif
c
c
c
      call vmov(xxadr,1,tempadr,1,nc)                   !! make Sxx cplx
      call vclr(xxadr,1,2*nc)
      call vmov(tempadr,1,xxadr,2,nc)
c
      call vmov(yyadr,1,tempadr,1,nc)                   !!make Syy cplx
      call vclr(yyadr,1,2*nc)
      call vmov(tempadr,1,yyadr,2,nc)
c
c
      call rfftsc(xyadr,fts,-3,0)                       !! pack for ifft
      call rfftsc(xxadr,fts,-3,0)
```

```
        call rfftsc(yyadr,fts,-3,0)
c
c
c
        call rfftb(xyadr,tempadr,fts,-1)                !! (un)biased ifft
        call vmul(tempadr,1,wadr,1,xyadr,1,fts)         !! in xyadr
c
        call rfftb(xxadr,tempadr,fts,-1)                !! (un)biased ifft
        call vmul(tempadr,1,wadr,1,xxadr,1,fts)         !! in xxadr
c
        call rfftb(yyadr,tempadr,fts,-1)                !! (un)biased ifft
        call vmul(tempadr,1,wadr,1,yyadr,1,fts)         !! in yyadr
c
c
c
c
        call apwr
        call apget(apbufr(1,1),xyadr,fts,2)             !! xyadr-->apbufr
        call apget(apbufr(1,2),xxadr,fts,2)             !! xxadr-->apbufr
        call apget(apbufr(1,3),yyadr,fts,2)             !! yyadr-->apbufr
        call apwd
c
c
c
c                                                       !! initalize output
        if (j.eq.1) then
                do k=1,3
                rtemp(k)=0.
                enddo
                scl(1)=sqrt(apbufr(1,2)*apbufr(1,3))
                if (nrm) scl(1)=float(fts)
                scl(2)=apbufr(1,2)
                scl(3)=apbufr(1,3)
        endif
c
c
        l=0                                             !! decimate S
        do 320 i=1,nc-1
        do k=1,3
        rtemp(k)=rtemp(k)+apbufs(i,k)
        enddo
        if (mod(i,8).eq.0) then
                l=l+1
                do k=1,3
                s(l,k)=10*log10(1.e-30+rtemp(k)*float(fts)/(8.*scl(k)))
                rtemp(k)=0.
                enddo
        endif
320     continue
c
c
c
c
        l=0
        do k=1,3
        rtemp(k)=0.
        enddo
c
```

```
        do 321 i-1,512                          !! fill Rxx,Ryy
c
        do k-2,3
        r(i,k)-apbufr(i,k)/scl(k)
        rtemp(k)-0.
        enddo
c
321     continue
c
c
c
c
        l-0
        do k-1,3
        rtemp(k)-0.
        enddo
c
        do 322 i-1,1024                         !! fill Rxy
c
        rtemp(1)-rtemp(1)+apbufr(i,1)**2
        rtemp(2)-rtemp(2)+apbufr(7168+i,1)**2
c
        if (mod(i,4).eq.0) then
                l-l+1
                r(256+l,1)-rtemp(1)/((scl(1)**2)*4.)
                r(l,1)-rtemp(2)/((scl(1)**2)*4.)
                rtemp(1)-0.
                rtemp(2)-0.
        endif
c
322     continue
c
c
c
c
        call wrec(rxych,r(1,1),4*j-3,4*j)             !! write r,s
        call wrec(sxych,s(1,1),4*j-3,4*j)
        call wrec(rxxch,r(1,2),4*j-3,4*j)
        call wrec(sxxch,s(1,2),4*j-3,4*j)
        call wrec(ryych,r(1,3),4*j-3,4*j)
        call wrec(syych,s(1,3),4*j-3,4*j)
c
c
        call otstr(':',0)
c
c
c
c
c
230     continue
c
c
        call cls(xch)                           !! close data files
        call cls(ych)
c
        call cls(rxych)                         !! close output files
```

```
        call cls(sxych)
        call cls(rxxch)
        call cls(sxxch)
        call cls(ryych)
        call cls(syych)
c
        call otstr(' ',1)
        call otstr('...files written',1)
c
c
c
c
        stop
        end
c
c
        include 'dra2:[abel.jsalib]jsaio.for/list'      !! get i/o
```

```
c       file i/o subroutines -- records
c       written as 512 byte blocks
c
c
c
c
c
        subroutine opn(ch,nam,oon)
c
c       opens file w/ name=nam on unit ch
c
        implicit none
        integer ch,ios
        character*(*) nam
        character*(3) oon
c                                               !! open file
c
        open(unit=ch,name=nam,status=oon,err=100,iostat=ios,
     x       recordtype='fixed',recordsize=512/4,access='direct',
     x       organization='sequential',blocksize=512*64)
        return
100     call prterr('opn error',ios)            !! print error
        return
        end
c
c
c
c
        subroutine cls(ch)
c
c       closes file on unit ch
c
        implicit none
        integer ch,ios
c
        close(unit=ch,status='keep',err=100)            !! close file
        return
c
100     call prterr('cls error',ios)                    !! print error
        return
        end
c
c
c
c
        subroutine rrecxy(chx,xbuf,chy,ybuf,sr,fr,xey)
c
c
c       reads starting sr finnishing fr:  chx-->xbuf, chy-->ybuf
c       if xey sets ybuf=xbuf -- no chy exsists
c
c
        implicit none
        integer chx,chy,nrcds,sr,fr,i
        byte xbuf(abs((fr-sr+1)*512)),ybuf(abs((fr-sr+1)*512))
        logical xey
```

```
c
c
      call rrec(chx,xbuf,sr,fr)               !! read x's records
c
      if (xey) then                           !! xey-->set y=x
              nrcds=fr-sr+1
              do 200 i=1,nrcds*512
200           ybuf(i)=xbuf(i)
      else
              call rrec(chy,ybuf,sr,fr)       !! else read y's records
      endif
c
      return
      end
c
c
c
c
c
      subroutine rrec(ch,buf,sr,fr)
c
c     reads fr-sr 512 byte records from ch to buf,  starting at
c     record sr finishing at fr
c
      implicit none
      integer sr,fr,ch,ios,i,s,j
      byte buf(abs((fr-sr+1)*512)),temp(512)
c
c
      if (sr .gt. fr) then            !! starting record after
              ios=999                 !! ending record
              goto 200
      endif
c
      do 100 i=sr,fr                  !! read records sr...fr
c
      read(unit=ch,rec=i,err=200,iostat=ios)temp
c
      s=(i-sr)*512                    !! put read buffer in
      do j=1,512                      !! return variable, buf
      buf(j+s)=temp(j)                !! 512 bytes/record
      enddo
c
100   continue
c
      return
c
200   call prterr('rrec err',ios)     !! print errors
      return
      end
c
c
c
c
c
      subroutine wrec(ch,buf,sr,fr)
c
c     writes buf to ch in 512 byte records --
```

```
c       starting at record sr, finnishing at fr
c
        implicit none
        integer s,ch,sr,fr,ios,i,j
        byte buf(abs((fr-sr+1)*512)),temp(512)


        if (sr .gt. fr) then            !! starting record after
                ios=999                 !! last record
                go to 200
        endif
c
        do 100 i=sr,fr                  !! write records sr...fr

        s=(i-sr)*512
        do j=1,512
        temp(j)=buf(j+s)                !! fill write buffer --
        enddo                           !! 512 bytes/record
c
100     write(unit=ch,rec=i,err=200,iostat=ios)temp
        return
c
200     call prterr('wrec error',ios)   !! print errors
        return
        end
c


c

c       terminal i/o routines

c

        subroutine otstr(str,nlf)
c
c       prints str, on the current line, followed by nlf cr/linefeeds


        implicit none
        integer nlf,l
        character*(*) str

        write(6,100)str                 !! prints string to terminal
100     format(1h+,a,$)

        if (nlf .lt. 1) return          !! prints lf's
        do 200 l=1,nlf
        write(6,300)
300     format(' ')
200     continue
        return
        end
c
```

```
c
c
c
      subroutine gtstr(pt,var,n)
c
c     gets a string from the terminal -- prompts w/ pt, var=string
c                                        var has length n<101
c
      implicit none
      character*(*) pt,var
      integer n,i,nchrs
      character*100 invar
c
      i=min(len(var),index(var,' '))
      write(6,100) pt,var                     !! write prompt, default
100   format(' ',a,' (',a<i>,'):',$)
c
      read(5,200)nchrs,(invar(i:i),i=1,nchrs) !! read srting
200   format(q,100a1)
c
      if (nchrs .eq. 0) return                !! return default
      do 300 i=1,n
      var(i:i)=' '
300   if (i .le. nchrs) var(i:i)=invar(i:i)   !! fill return variable
      return
      end
c
c
      subroutine gtint(pt,igr,mx,mn)
c
c     gets integer from terminal -- prompts user w/ pt, accepts
c                                   igr in range (mn,mx)
c
c
c
      implicit none
      character*10 itstr
      integer igr,mx,mn,it,nchrs,i,lmn,lmx
      character*(*)pt
c
      i=2+max(1,int(log10(abs(float(igr))+.1)))
      write(6,100)pt,igr                      !! write prompt, default
100   format(' ',a,' (',i<i>,'):',$)
c
200   read(5,400)nchrs,(itstr(i:i),i=1,10)    !! get string
400   format(q,10a1)
c
      if (nchrs .eq. 0) return                !! take default if no
c                                             !! entry
      call s2i(it,itstr)                      !! convert str-->int
c
      if ((it .le. mx) .and. (it .ge. mn)) then       !! check range
              igr=it
              return
      endif
      lmn=2+max(1,int(log10(abs(floatj(mn))+.1)))
      lmx=2+max(1,int(log10(abs(floatj(mx))+.1)))
      write(6,300)mn,mx                               !! not in range
```

```
300     format(' The (min,max)are (',i<lmn>,',',i<lmx>,') -- reenter:',$)
        go to 200
        end
c
c
c
        subroutine s2i(itr,str)
c
c       converts string -- str -- to integer -- itr.
c
        implicit none
        integer a,l
        integer itr,i,j,temp,pt
        character*(*) str
c
        temp=0
        pt=1
        l=len(str)
c
        do 100 i=0,l-1                    !! itr=sum(str(i)*10**i)
        a=ichar(str(l-i:l-i))
        if (a .eq. 48) go to 200                          !! a='0'
        if (a .eq. 45) go to 300                          !! a='-'
        if ((a .lt. 49) .or. (a .gt. 57)) go to 100       !! a not in 1--9
        temp=temp+pt*(a-48)
200     pt=pt*10
100     continue
        itr=temp
        return
c
300     itr=-temp
        return
        end
c
c
c
        subroutine gtrel(pt,rel,mx,mn)
c
c       gets real*4 from terminal -- prompts user w/ pt, accepts
c                                        rel in range (mn,mx)
c
c
c
        implicit none
        character*10 rlstr
        real*4 rel,mx,mn,rl
        integer nchrs,i,lmx,lmn
        character*(*)pt
c
        write(6,100)pt,rel                        !! write prompt, default
100     format(' ',a,' (',f<5+max(1,int(log10(abs(rel)+.1)))>.3,'):',$)
c
200     read(5,400)nchrs,(rlstr(i:i),i=1,10)     !! get string
400     format(q,10a1)
c
        if (nchrs .eq. 0) return                  !! take default if no
```

```
c                                                   !! entry
        call s2r(rl,rlstr)                          !! convert str-->int
c
        if ((rl .le. mx) .and. (rl .ge. mn)) then          !! check range
                rel=rl
                return
        endif
        lmx=5+max(1,int(log10(abs(mx)+.1)))
        lmn=5+max(1,int(log10(abs(mn)+.1)))
        write(6,300)mn,mx                                  !! not in range
300     format(' The (min,max)are (',f<lmn>.3,',',f<lmx>.3,')
     x    -- reenter:',$)
        go to 200
        end
c
c
c
        subroutine s2r(r,str)
c
c       converts string -- str -- to real -- r.
c
        implicit none
        integer a,l,i,j
        real*4 r,temp,pt
        character*(*) str
c
        temp=0.
        pt=1.
        l=len(str)
c
        do 100 i=0,l-1                                 !! itr=sum(str(i)*10**i)
        a=ichar(str(l-i:l-i))
        if (a .eq. 45) go to 300                       !! a='-'
        if (a .eq. 48) go to 200                       !! a='0'
        if (a .eq. 46) then                            !! a='.'
                temp=temp/pt
                pt=1.
                go to 100
        endif
        if ((a .lt. 49) .or. (a .gt. 57)) go to 100    !! a not in 1--9
        temp=temp+pt*float(a-48)
200     pt=pt*10.
100     continue
        r=temp
c
        return
c
300     r=-temp
c
c
        return
        end
c
c
c
c
        subroutine clrscrn
c
```

```
c       clears the screen
c
        byte sbuf(4)
c
        sbuf(1)=27 ! ESC
        sbuf(2)=91 ! [
        sbuf(3)=50 ! 2
        sbuf(4)=74 ! J
        write(6,100)(sbuf(i),i=1,4)
100     format(1h+,4a1)
c
        return
        end
c
c


        logical function gtyes(pt)

        prompts user w/ pt, returns .true. if 'yes'
c
        implicit none
        character ystr
        character*(*) pt
c
        gtyes=.false.
        write(6,100) pt                          !! write prompt
100     format(' ',a,$)
        read(5,200) ystr                         !! get response
200     format(a1)
        if ((ystr .eq. 'y') .or. (ystr .eq.'Y')) gtyes=.true.
        return                                   !! true if first
        end                                      !! character is y or Y
c



        subroutine prterr(msg,ios)

        prints runtime error codes
c
        character*(*) msg
        integer ios
c
        write(6,100) msg,ios                     !! write message and code
100     format(' ',a,':',i4)                     !! to terminal

        return
        end
```

APPENDIX 4

ADEC LINETRACKER SOFTWARE

```
        integer*4       number_of_points
        integer*4       output_file_number
        integer*4       starting_point
        character*20    temp_string
        character*80    oname,iname,tfnam
        integer*4       och(4),ich
        logical         gtyes
c
c
c       initialization
c
c

        do i=1,4
        och(i)=14+i
        enddo
        ich=2
        iname='data:rxytest.r4'
        oname='data:rxygm.r4'
        tfnam='data:track.fil'
        number_of_points=100
        sl=1
        nbins=512
        sbin=1
        scl=1.
        thr=2.
        pct=.7
c
c       input
c
100     call gtstr('Enter the output gram file name',oname,80)
        call gtstr('Enter the output track file name',tfnam,80)
        call gtstr('Enter the input file name',iname,80)
        call gtint('Enter the number of epochs',number_of_points,700,1)
        call gtint('Enter the starting epoch number',sl,9999,1)
        call gtint('Enter the number of fft bins',nbins,99999,1)
        call gtint('Enter the starting bin number',sbin,99999,1)
        call gtrel('Enter the input scale factor',scl,99999.,.00001)
        call gtrel('Enter the pw threshold',thr,99.,1.)
        call gtrel('Enter the pw percent',pct,1.,0.)
        call otstr(' ',2)
        if (.not.gtyes('Inputs ok (y/n): ')) go to 100
        call otstr(' ',5)

        call opn(och,oname,'new')
        call opn(ich,iname,'old')
        call adecini(number_of_points)
c
c       body
c
c

        starting_point = 1

        call lfcr
```

```
        call percom ( 0, number_of_points, 5)

        do i = 1, number_of_points

            call percom ( i, number_of_points, 5)
c
c
            call adecrd ( ich,i,fft_amplitudes,sl,scl,nbins,sbin)
c
c                                            !! read in an FFT line and normalize
c
            starting_point = starting_point + 512

            call doadec ( fft_amplitudes) ! process FFT line

            if ( mod ( i, 175) .eq. 1 .and. i .ne. 1) then
                call abelfin ( och,gram_number, gram)
                do j = 1, 89600
                    gram ( j) = 0
                end do
                gram_number = gram_number + 1
                temp_string = 'adec' // char ( gram_number + 48) // '.grm'
                call opn(och(gram_number),temp_string,'new')
            end if

            call adecgram ( mod ( i - 1, 175) + 1, gram) ! output adec results

        call writestuff(i,fft_amplitudes,thr,pct,number_of_points,tfnam)

        end do

        call abelfin ( och, gram_number, gram)
        call cls(ich)
        end
c
        include 'jsaio.for'                       !! get jsa's i/o
c
c
c
c
c
        subroutine abelfin(ch,gn,x)
c
c
c       writes gram -- x to channel ch(gn)
c
c
        implicit none
        integer*4       ch(4),gn,i,j
        real            temp(512),x(512,175)
c
c
c
        do i=1,512
        temp(i)=0
        enddo
c
c
```

```
        do 1000 i=1,175
c
        do 2000 j=1,512
        temp(j)=x(j,i)
 2000   continue
c
        call wrec(ch(gn),temp,4*(i-1)+1,4*i)
 1000   continue
c


        call cls(ch(gn))
        end
        subroutine assoc ( association_threshold, association_gate_min,
        1    association_gate_max, association_gate_slope, fft_amplitude,
        1    fft_associated, max_fft_bin_number)

        implicit none

        real            association_gate_max
        real            association_gate_min ( 1)
        real            association_gate_slope ( 1)
        real            association_threshold
        real            fft_amplitude ( 1)
        logical*1       fft_associated ( 1)
        integer         max_fft_bin_number

        real            association_gate
        integer         bin
        logical*1       lower_edt
        real            lower_limit
        integer         lower_limit_bin
        integer         other_track
        integer         track
        logical*1       upper_edt
        real            upper_limit
        integer         upper_limit_bin

        include         'tracks.inc'
```

```
      do bin = 1, max_fft_bin_number
          fft_associated ( bin) = .false.
      end do

      do track = 1, number_of_tracks

          if ( t_adaptive_amplitude ( track) .gt. association_threshold) then

              association_gate =
      1           association_gate_min ( t_type ( track)) +
      1           association_gate_slope ( t_type ( track)) *
      1           ( t_adaptive_amplitude ( track) - association_threshold)

              association_gate =
      1           min ( association_gate, association_gate_max)

          else

              association_gate = association_gate_min ( t_type ( track))

          end if

          upper_limit = min ( t_frequency ( track) + association_gate,
      1       float ( max_fft_bin_number))
          lower_limit = max ( t_frequency ( track) - association_gate, 1.)

c         Check for to see if any higher frequency edt track will affect the
c         upper limit of the association.

          do other_track = track + 1, number_of_tracks
              if ( t_type ( other_track) .eq. edt) then
                  upper_edt = .true.
                  goto 1000
              end if
          end do

          upper_edt = .false.

1000      continue

          if ( upper_edt) then
              upper_limit_bin = min ( upper_limit,
      1           ( t_frequency ( track) + t_frequency ( other_track)) * .5)
      1           + 0.5
          else
              upper_limit_bin = upper_limit + .5
          end if

c         Check to see if any lower frequency edt track will affect
c         the lower limit of the association

          do other_track = track - 1, 1, -1
              if ( t_type ( other_track) .eq. edt) then
                  lower_edt = .true.
                  goto 2000
              end if
          end do
```

```
        lower_edt = .false.

000     continue

        if ( lower_edt) then
            lower_limit_bin = max ( lower_limit,
     1          ( t_frequency ( track) + t_frequency ( other_track)) * .5)
     1          + .5
        else
            lower_limit_bin = lower_limit + .5
        end if
```

```
c           Associate cell(s) with track.

            t_associated_amplitude ( track) = fft_amplitude ( lower_limit_bin)
            t_associated_cell ( track) = lower_limit_bin
            do bin = lower_limit_bin, upper_limit_bin
                if ( fft_amplitude ( bin) .gt. t_associated_amplitude ( track))
        1           then

                    t_associated_amplitude ( track) = fft_amplitude ( bin)
                    t_associated_cell ( track) = bin

                end if
                if ( t_type ( track) .eq. edt) fft_associated ( bin) = .true.
                    ! all tracks in window of edt are associated
            end do

            fft_associated ( t_associated_cell ( track)) = .true.

            if ( t_associated_amplitude ( track) .lt.
        1       t_adaptive_amplitude ( track)) then

                t_dcell ( track) = ( t_associated_cell ( track) -
        1           t_frequency ( track)) * ( t_associated_amplitude ( track) /
        1           t_adaptive_amplitude ( track))

            else

                t_dcell ( track) = t_associated_cell ( track) -
        1           t_frequency ( track)

            end if


        end do

        end
        real function besseli0 ( x)

        real            x

        real            arg
        real            b
        integer         i
        real            term


        b = 1
        arg = x * x * .25
        term = 1

        do i = 1, 25

            term = term * arg / float ( i) ** 2
            if ( abs ( term / b) .lt. 1e-4) goto 1000
            b = b + term

        end do
```

```
1000    besseli0 = b

        end
        subroutine cmlne ( window_size, upper, lower, ns, rx, x)

        implicit complex*8 ( a - z)

        integer*4       window_size
        integer*4       upper
        integer*4       lower
        integer*4       ns
        real            rx ( *)
        real            x ( *)

        integer*4       i
        integer*4       j
        real            list ( 64)
        real            mean
        real            temp

c       Build first list

        do i = 1, window_size
            list ( i) = rx ( i)
        end do

c       Sort list ( crudely right now.)

        do i = 1, window_size
            do j = i + 1, window_size
                if ( list ( i) .gt. list ( j)) then
                    temp = list ( j)
                    list ( j) = list ( i)
                    list ( i) = temp
                end if
            end do
        end do

        Calculate mean

        mean = 0.
        do i = lower, upper
            mean = mean + list ( i)
        end do
        mean = mean / float ( upper - lower + 1)

        Calculate noise estimate for left end of data

        do i = 1, window_size / 2
            x ( i) = mean
        end do

        Calculate noise estimate for middle of data

        do i = window_size / 2 + 1, ns - window_size / 2
            call sorto ( list, window_size, rx ( i - window_size / 2))
```

```
            call sorti ( list, window_size, rx ( i + window_size / 2))
            call aver ( list, upper, lower, mean)
            x ( i) = mean
        end do

c       Calculate noise estimate for right end of data

        do i = ns - window_size / 2 + 1, ns
            x ( i) = mean
        end do

        end
```

```
      block data

      include         'tracks.inc'

      data free_entry / 1/
      data number_of_tracks / 0/

      end
      subroutine adecfin ( output_file_number, gram)

      implicit        none

      real            gram ( 58100)
      integer*4       output_file_number

      real            buffer ( 512)
      integer         buffer_pointer
      integer         i
      integer         j
      integer*4       page

      page = 0
      buffer_pointer = 1

      do i = 1, 332
          do j = 1, 175

              buffer ( buffer_pointer) = gram (( j - 1) * 332 + i)
              if ( buffer_pointer .eq. 512) then
                  call ptpage ( page, output_file_number, buffer)
                  page = page + 1
                  buffer_pointer = 0
              end if

              buffer_pointer = buffer_pointer + 1

          end do
      end do

c     Zero pad last page.

      do i = buffer_pointer, 512
          buffer ( i) = 0.
      end do

      call ptpage ( page, output_file_number, buffer)
      call nclose ( output_file_number)

      end
      subroutine adecgram ( i, gram)

      implicit        none

      real            gram ( 512, 175)
      integer         i

      integer         bin
      integer         track
```

```
include         'tracks.inc'

do bin = 1, 512
    gram ( bin, i) = 0.
end do

do track = 1, number_of_tracks
    bin = t_frequency ( track) + .5
    if ( t_detection_flag ( track)) then
        gram ( bin, i) = 7.
    else
        gram ( bin, i) = 2.
    end if
end do

end
subroutine adecini(number_of_points)

implicit        none

integer*4       number_of_points
character*80    coefficient_file_name
logical*1       error
integer         i
real            new_track_pfa
real            old_track_pd
real            old_track_pfa
character*80    output_file_name
character*80    param_file_name
integer*4       param_unit
integer*4       range ( 2)
real            temp_real

include         'jhjlib.inc'
include         'param.inc'
```

```
        call getjfn ( param_unit)
        error = .false.
100     param_file_name = 'adec.prm'
        if ( error) call outmes ( 'Couln''t find file.  Try again.')
        error = .true.
        call askstr ( 'Adec parameter file', param_file_name)
        open ( unit = param_unit, err = 100, type = 'old', readonly,
        1    form = 'formatted', file = param_file_name)

        Read in adec paramters

        type 1000

        read ( param_unit, *) amplitude_smoothing_constant
        type 1001, 'amplitude_smoothing_constant', amplitude_smoothing_constant
        read ( param_unit, *) association_gate_min ( 1)
        type 1001, 'association_gate_min', association_gate_min ( 1)
        read ( param_unit, *) association_gate_max
        type 1001, 'association_gate_max', association_gate_max
        do i = 1, 3
            read ( param_unit, *) association_gate_slope ( i)

            type 1001, 'association_gate_slope ' // char (i + 48),
        1       association_gate_slope ( i)
        end do
        read ( param_unit, *) association_threshold
        type 1001, 'association_threshold', association_threshold
        read ( param_unit, *) old_track_pd
        type 1001, 'old_track_pd', old_track_pd
        read ( param_unit, *) old_track_pfa
        type 1001, 'old_track_pfa', old_track_pfa
        detection_threshold = log ( old_track_pd / old_track_pfa)
        drop_threshold = log (( 1. - old_track_pd) / ( 1. - old_track_pfa))
        read ( param_unit, *) temp_real
        display_threshold = detection_threshold + temp_real
        type 1001, 'display_threshold', display_threshold
        read ( param_unit, *) dynamic_tracking_threshold
        type 1001, 'dynamic_tracking_threshold', dynamic_tracking_threshold
        read ( param_unit, *) faca
        type 1001, 'faca', faca
        read ( param_unit, *) log_likelihood_max
        type 1001, 'log_likelihood_max', log_likelihood_max
        read ( param_unit, *) max_fft_bin_number
        type 1002, 'max_fft_bin_number', max_fft_bin_number
        read ( param_unit, *) merge_gate
        type 1001, 'merge_gate', merge_gate
        read ( param_unit, *) rate_gate
        type 1001, 'rate_gate', rate_gate
        read ( param_unit, *) new_track_pfa
        type 1001, 'new_track_pfa', new_track_pfa

        close ( unit = param_unit)

        type 1003

        do i = 1, 3
            new_track_threshold ( i) = sqrt ( -2. * log ( new_track_pfa))
        end do
```

```
        association_gate_min ( 2) = association_gate_min ( 1)
        association_gate_min ( 3) = association_gate_min ( 1)

        detection_threshold = log ( old_track_pd / old_track_pfa)
        drop_threshold = log (( 1. - old_track_pd) / ( 1. - old_track_pfa))

1000    format ( '0Adec parameters:'//)
1001    format ( 1x, a, '=', g13.6)
1002    format ( 1x, a, '=', i13)
1003    format ( '0')

        end
        subroutine adecrd ( ch, starting_point,
        1     fft_amplitudes, sl, scl,nbins,sbin)

        real            fft_amplitudes ( *),scl
        integer*4       ch,sl,i,j,k,nbins,sbin
        integer*4       starting_point

        complex         coefficient_array ( 512)
        real            nse ( 512)
        real            temp ( 512)
        real            temp2 ( 9999)

        include         'jhjlib.inc'
c
c
c
        j=(sl+starting_point-2)*nbins+sbin
        k=int(float(j)/128)+1
        l=k+int(float(nbins)/128)+1
        call rrec(ch,temp2,k,l)

        ii=j-(k-1)*128+1

        do i=1,512
        temp(i)=scl*temp2(i+ii)
        enddo
c
c
c
        call cmlne ( 32, 16, 1, 512, temp, nse)

        do i = 1, 512
            fft_amplitudes ( i) = scl*temp2 ( i+ii) / sqrt(nse(i))*.558
        end do

        end
        program adecw

        implicit none

        integer*4       coefficient_file_number
        real            fft_amplitudes ( 512), scl, thr, pct
        real            gram ( 89600)
        integer         gram_number / 1/
        integer         i,j
        integer*4       sl,sbin,nbins
```

```
      subroutine sorto ( list, window_size, value)

      real            list ( 1)
      integer*4       window_size
      real            value

      integer*4       i
      integer*4       j

      do i = 1, window_size
          if ( value .eq. list ( i)) goto 100
      end do

100   continue

c     Compress list

      do j = i, window_size - 1
          list ( j) = list ( j + 1)
      end do

      end
```

```
        subroutine sorti ( list, window_size, value)

        real            list ( 1)
        integer*4       window_size
        real            value

        integer*4       i
        integer*4       j
        real            temp1
        real            temp2

        do i = 1, window_size - 1
            if ( value .lt. list ( i)) then
                goto 100
            end if
        end do

        i = window_size

100     continue

        temp2 = value

c       Compress list

        do j = i, window_size
            temp1 = list ( j)
            list ( j) = temp2
            temp2 = temp1
        end do

        end
```

```
      subroutine aver ( list, upper, lower, mean)

      real            list ( 1)
      integer*4       upper
      integer*4       lower
      real            mean

      mean = 0.

      do i = lower, upper
          mean = mean + list ( i)
      end do

      mean = mean / float ( upper - lower + 1)

      end
      subroutine compress

      implicit none

      integer         target_track
      integer         track
      integer         track_counter

      include         'tracks.inc'

      track_counter = 0
      target_track = 1

      do track = 1, free_entry - 1

          if ( t_frequency ( track) .ge. 0) then

              track_counter = track_counter + 1
              if ( track .ne. target_track) then

                  t_adaptive_amplitude ( target_track) =
     1                t_adaptive_amplitude ( track)
                  t_age ( target_track) = t_age ( track)
                  t_associated_amplitude ( target_track) =
     1                t_associated_amplitude ( track)
                  t_associated_cell ( target_track) =
     1                t_associated_cell ( track)
                  t_dcell ( target_track) = t_dcell ( track)
                  t_detection_flag ( target_track) =
     1                t_detection_flag ( track)
                  t_frequency ( target_track) = t_frequency ( track)
                  t_frequency_rate ( target_track) =
     1                t_frequency_rate ( track)
                  t_id ( target_track) = t_id ( track)
                  t_log_likelihood ( target_track) =
     1                t_log_likelihood ( track)
                  t_type ( target_track) = t_type ( track)

              end if

              target_track = target_track + 1
```

```
            end if

        end do

        number_of_tracks = track_counter
        free_entry = number_of_tracks + 1

        end
        subroutine deltra ( track)

        implicit none

        integer         track

        include         'tracks.inc'

        t_frequency ( track) = -1.0

        end
        subroutine doader ( fft_amplitudes)

        implicit none

        real            fft_amplitudes ( *)

        real            alpha ( 32) / 10 * .18750, .20099, .21301, .22601,
        1                   .23700, .24899, .26099, .28125, .30600, .32999,
        1                   .36499, 12 * .37900/

        real            beta ( 32) / 9 * .87500, .01941, .02246, .02637,
        1                   .02863, .03186, .03640, .03918, .04602, .05527,
        1                   .06522, .07661, 12 * .08862/

        logical*1       fft_associated ( 512)

        include         'param.inc'
        include         'tracks.inc'
```

```
c       Associate existing tracks with FFT cells.

        call assoc ( association_threshold, association_gate_min,
     1     association_gate_max, association_gate_slope, fft_amplitudes,
     1     fft_associated, max_fft_bin_number)

        Update existing tracks.

        call update ( amplitude_smoothing_constant, detection_threshold,
     1     display_threshold, drop_threshold, log_likelihood_max)

c       Create new tracks.

        call newtra ( fft_amplitudes, fft_associated, max_fft_bin_number,
     1     new_track_threshold)

        Put the track table back into order of increasing frequency.

        call sort

        Merge equivalent tracks.

        call merge ( merge_gate, rate_gate)

        Smooth and predict new frequency and rate.

        call smooth ( dynamic_tracking_threshold, faca, alpha, beta,
     1     max_fft_bin_number)

        end
        subroutine domerg ( track1, track2)

        implicit none

        integer         track1
        integer         track2

        integer         deleted_track
        integer         retained_track

        include         'tracks.inc'
```

```
      if (( t_type ( track1) .eq. edt) .and. ( t_type ( track2) .eq. edt))
     1    then

          if ( t_detection_flag ( track1) .eq. t_detection_flag ( track2))
     1        then

              if ( t_adaptive_amplitude ( track1) .ge.
     1            t_adaptive_amplitude ( track2)) then

                  retained_track = track1

              else

                  retained_track = track2

              end if

          else

              if ( t_detection_flag ( track1)) then

                  retained_track = track1

              else

                  retained_track = track2

              end if

          end if

      else if ( t_type ( track1) .eq. edt) then

          retained_track = track1

      else if ( t_type ( track2) .eq. edt) then

          retained_track = track2

      else

          if ( t_log_likelihood ( track1) .gt. t_log_likelihood ( track2))
     1        then

              retained_track = track1

          else if ( t_log_likelihood ( track1) .lt.
     1        t_log_likelihood ( track2)) then

              retained_track = track2

          else if ( t_adaptive_amplitude ( track1) .gt.
     1        t_adaptive_amplitude ( track2)) then

              retained_track = track1

          else
```

```
                retained_track = track2

            end if

        end if

        if ( retained_track .eq. track1) then
            deleted_track = track2
        else
            deleted_track = track1
        end if

        if ( t_age ( track1) .ge. t_age ( track2)) then
            t_id ( retained_track) = t_id ( track1)
        else
            t_id ( retained_track) = t_id ( track2)
        end if

        t_age ( retained_track) = max ( t_age ( track1), t_age ( track2))

        call deltra ( deleted_track)

        end
        file i/o subroutines -- records
c       written as 512 byte blocks



c
c

        subroutine opn(ch,nam,oon)
c
        opens file w/ name=nam on unit ch

        implicit none
        integer ch,ios
        character*(*) nam
        character*(3) oon
c                                               !! open file

        open(unit=ch,name=nam,status=oon,err=100,iostat=ios,
     x       recordtype='fixed',recordsize=512/4,access='direct',
     x       organization='sequential',blocksize=512*64)
        return
100     call prterr('opn error',ios)            !! print error
        return
        end

c



        subroutine cls(ch)

        closes file on unit ch
c
        implicit none
```

```
        integer ch,ios
c
        close(unit=ch,status='keep',err=100)          !! close file
        return
c
100     call prterr('cls error',ios)                  !! print error
        return
        end
c
c
c
c
c
        subroutine rrecxy(chx,xbuf,chy,ybuf,sr,fr,xey)
c
c
c       reads starting sr finnishing fr:  chx-->xbuf, chy-->ybuf
c       if xey sets ybuf=xbuf -- no chy exsists
c
c
        implicit none
        integer chx,chy,nrcds,sr,fr,i
        byte xbuf(abs((fr-sr+1)*512)),ybuf(abs((fr-sr+1)*512))
        logical xey
c
c
        call rrec(chx,xbuf,sr,fr)                !! read x's records
c
        if (xey) then                            !! xey-->set y=x
                nrcds=fr-sr+1
                do 200 i=1,nrcds*512
200             ybuf(i)=xbuf(i)
        else
                call rrec(chy,ybuf,sr,fr)        !! else read y's records
        endif
c
        return
        end
c
c
c
c
c
        subroutine rrec(ch,buf,sr,fr)
c
c       reads fr-sr 512 byte records from ch to buf,  starting at
c       record sr finishing at fr
c
        implicit none
        integer sr,fr,ch,ios,i,s,j
        byte buf(abs((fr-sr+1)*512)),temp(512)
c
c
        if (sr .gt. fr) then             !! starting record after
                ios=999                  !! ending record
                goto 200
        endif
c
```

```
        do 100 i=sr,fr                      !! read records sr...fr
c
        read(unit=ch,rec=i,err=200,iostat=ios)temp

        s=(i-sr)*512                        !! put read buffer in
        do j=1,512                          !! return variable, buf
        buf(j+s)=temp(j)                    !! 512 bytes/record
        enddo
c
100     continue

        return

200     call prterr('rrec err',ios)         !! print errors
        return
        end
c
c
c



        subroutine wrec(ch,buf,sr,fr)
c
c       writes buf to ch in 512 byte records --
c       starting at record sr, finnishing at fr
c
        implicit none
        integer s,ch,sr,fr,ios,i,j
        byte buf(abs((fr-sr+1)*512)),temp(512)


        if (sr .gt. fr) then                !! starting record after
                ios=999                     !! last record
                go to 200
        endif
c
        do 100 i=sr,fr                      !! write records sr...fr

        s=(i-sr)*512
        do j=1,512
        temp(j)=buf(j+s)                    !! fill write buffer --
        enddo                               !! 512 bytes/record
c
100     write(unit=ch,rec=i,err=200,iostat=ios)temp
        return
c
200     call prterr('wrec error',ios)       !! print errors
        return
        end
c
c
c
c
c       terminal i/o routines
c
```

```
c
c
c
      subroutine otstr(str,nlf)
c
c     prints str, on the current line, followed by nlf cr/linefeeds
c
c
      implicit none
      integer nlf,l
      character*(*) str
c
      write(6,100)str                    !! prints string to terminal
100   format(1h+,a,$)
c
      if (nlf .lt. 1) return             !! prints lf's
      do 200 l=1,nlf
      write(6,300)
300   format(' ')
200   continue
      return
      end
c
c
c
c
c
      subroutine gtstr(pt,var,n)
c
c     gets a string from the terminal -- prompts w/ pt, var=string
c                                        var has length n<101
c
      implicit none
      character*(*) pt,var
      integer n,i,nchrs
      character*100 invar
c
      i=min(len(var),index(var,' '))
      write(6,100) pt,var                          !! write prompt, default
100   format(' ',a,' (',a<i>,'):',$)
c
      read(5,200)nchrs,(invar(i:i),i=1,nchrs) !! read srting
200   format(q,100a1)
c
      if (nchrs .eq. 0) return                     !! return default
      do 300 i=1,n
      var(i:i)=' '
300   if (i .le. nchrs) var(i:i)=invar(i:i)   !! fill return variable
      return
      end
c
c
      subroutine gtint(pt,igr,mx,mn)
c
c     gets integer from terminal -- prompts user w/ pt, accepts
c                                   igr in range (mn,mx)
c
c
```

```
c
        implicit none
        character*10 itstr
        integer igr,mx,mn,it,nchrs,i,lmn,lmx
        character*(*)pt

        i=2+max(1,int(log10(abs(float(igr))+.1)))
        write(6,100)pt,igr                          !! write prompt, default
100     format(' ',a,' (',i<i>,'):',$)

200     read(5,400)nchrs,(itstr(i:i),i=1,10)     !! get string
400     format(q,10a1)

        if (nchrs .eq. 0) return                    !! take default if no
c                                                   !! entry
        call s2i(it,itstr)                          !! convert str-->int

        if ((it .le. mx) .and. (it .ge. mn)) then        !! check range
                igr=it
                return
        endif
        lmn=2+max(1,int(log10(abs(floatj(mn))+.1)))
        lmx=2+max(1,int(log10(abs(floatj(mx))+.1)))
        write(6,300)mn,mx                                !! not in range
300     format(' The (min,max)are (',i<lmn>,',',i<lmx>,') -- reenter:',$)
        go to 200
        end
c
c
        subroutine s2i(itr,str)

c       converts string -- str -- to integer -- itr.

        implicit none
        integer a,l
        integer itr,i,j,temp,pt
        character*(*) str

        temp=0
        pt=1
        l=len(str)
c
        do 100 i=0,l-1                   !! itr=sum(str(i)*10**i)
        a=ichar(str(l-i:l-i))
        if (a .eq. 48) go to 200                     !! a='0'
        if (a .eq. 45) go to 300                     !! a='-'
        if ((a .lt. 49) .or. (a .gt. 57)) go to 100  !! a not in 1--9
        temp=temp+pt*(a-48)
200     pt=pt*10
100     continue
        itr=temp
        return
c
300     itr=-temp
        return
        end
```

```
c
c
c
c
      subroutine gtrel(pt,rel,mx,mn)
c
c     gets real*4 from terminal -- prompts user w/ pt, accepts
c                                  rel in range (mn,mx)
c
c
c
      implicit none
      character*10 rlstr
      real*4 rel,mx,mn,rl
      integer nchrs,i,lmx,lmn
      character*(*)pt
c
      write(6,100)pt,rel                        !! write prompt, default
100   format(' ',a,' (',f<5+max(1,int(log10(abs(rel)+.1)))>.3,'):',$)
c
200   read(5,400)nchrs,(rlstr(i:i),i=1,10)     !! get string
400   format(q,10a1)
c
      if (nchrs .eq. 0) return                  !! take default if no
c                                               !! entry
      call s2r(rl,rlstr)                        !! convert str-->int
c
      if ((rl .le. mx) .and. (rl .ge. mn)) then       !! check range
              rel=rl
              return
      endif
      lmx=5+max(1,int(log10(abs(mx)+.1)))
      lmn=5+max(1,int(log10(abs(mn)+.1)))
      write(6,300)mn,mx                               !! not in range
300   format(' The (min,max)are (',f<lmn>.3,',',f<lmx>.3,')
     x  -- reenter:',$)
      go to 200
      end
c
c
c
      subroutine s2r(r,str)
c
c     converts string -- str -- to real -- r.
c
      implicit none
      integer a,l,i,j
      real*4 r,temp,pt
      character*(*) str
c
      temp=0.
      pt=1.
      l=len(str)
c
      do 100 i=0,l-1                                 !! itr=sum(str(i)*10**i)
      a=ichar(str(l-i:l-i))
      if (a .eq. 45) go to 300                       !! a='-'
      if (a .eq. 48) go to 200                       !! a='0'
```

```
        if (a .eq. 46) then                            !! a='.'
                temp=temp/pt
                pt=1.
                go to 100
        endif
        if ((a .lt. 49) .or. (a .gt. 57)) go to 100  !! a not in 1--9
        temp=temp+pt*float(a-48)
200     pt=pt*10.
100     continue
        r=temp
c
        return
c
300     r=-temp
c
c
        return
        end
c


c
        subroutine clrscrn

        clears the screen
c
        byte sbuf(4)

        sbuf(1)=27 ! ESC
        sbuf(2)=91 ! [
        sbuf(3)=50 ! 2
        sbuf(4)=74 ! J
        write(6,100)(sbuf(i),i=1,4)
100     format(1h+,4a1)

        return
        end


c



        logical function gtyes(pt)

        prompts user w/ pt, returns .true. if 'yes'

        implicit none
        character ystr
        character*(*) pt
c
        gtyes=.false.
        write(6,100) pt                          !! write prompt
100     format(' ',a,$)
        read(5,200) ystr                         !! get response
200     format(a1)
        if ((ystr .eq. 'y') .or. (ystr .eq.'Y')) gtyes=.true.
        return                                   !. true if first
        end                                      !! character is y or Y
```

```
c
c
c
c
c
      subroutine prterr(msg,ios)
c
c     prints runtime error codes
c
      character*(*) msg
      integer ios
c
      write(6,100) msg,ios                    !! write message and code
100   format(' ',a,':',i4)                    !! to terminal
c
      return
      end

      real function log_likelihood_ratio ( type, amplitude)

      real            amplitude
      integer         type

      real            besseli0

c     Adec log likelihood ratio.  Pfa - 1e-4, Pd - .5

      log_likelihood_ratio - log ( besseli0 ( amplitude)) - .5

      end
      subroutine maktra ( bin, amplitude, type)

      implicit none

      real            amplitude
      integer         bin
      real            log_likelihood_ratio
      integer         type

      integer         id / 0/

      include         'tracks.inc'

      t_frequency ( free_entry) - bin
      t_frequency_rate ( free_entry) - 0.
      t_log_likelihood ( free_entry) -
     1   log_likelihood_ratio ( type, amplitude)
      t_adaptive_amplitude ( free_entry) - amplitude
      t_age ( free_entry) - 0
      t_dcell ( free_entry) - 0
      t_type ( free_entry) - type
      t_detection_flag ( free_entry) - 0
      t_id ( free_entry) - id

      id - id + 1
      free_entry - free_entry + 1

      end
```

```
      subroutine merge ( merge_gate, rate_gate)

      implicit none

      real             merge_gate
      real             rate_gate

      integer          other_track
      integer          track

      include          'tracks.inc'
```

```
        do track = 2, number_of_tracks

            do other_track = track - 1, 1, -1

                if ( t_frequency ( track) .lt. 0) goto 1000

                if ( t_frequency ( other_track) .lt. 0) then ! deleted track

                else if ((( t_type ( track) .eq. weak) .and.
        1           ( t_type ( other_track) .eq. strong)) .or.
        1           (( t_type ( track) .eq. strong) .and.
        1           ( t_type ( other_track) .eq. weak))) then

                else if ( t_frequency ( track) - t_frequency ( other_track)
        1           .gt. merge_gate) then

                    goto 1000

                else if (( t_type ( track) .eq. edt) .and.
        1           ( t_type ( other_track) .eq. edt)) then

                    if (( t_age ( track) .eq. 1) .or.
        1               ( t_age ( other_track) .eq. 1)) then

                        call domerg ( track, other_track)

                    else if ( abs ( t_frequency_rate ( track) -
        1               t_frequency_rate ( other_track)) .le. rate_gate) then

                        call domerg ( track, other_track)

                    end if

                else

                    call domerg ( track, other_track)

                end if

            end do

1000        continue

        end do

        call compress

        end
        subroutine newtra ( fft_amplitude, fft_associated, max_fft_bin_number,
        1    threshold)

        implicit none

        real            fft_amplitude ( 1)
        logical*1       fft_associated ( 1)
        integer         max_fft_bin_number
        real            threshold ( 1)
```

```
      integer           bin
      logical*1         looking_for_downslope

      include           'tracks.inc'
```

```
      looking_for_downslope = .false.

      do bin = 1, max_fft_bin_number - 1

          if ( fft_amplitude ( bin) - fft_amplitude ( bin + 1) .gt. 0)
      1       then

              if ( looking_for_downslope) then

                  if ( fft_amplitude ( bin) .lt. threshold ( weak)) then

                  else if ( fft_associated ( bin)) then

                  else if ( fft_amplitude ( bin) .gt. threshold ( strong))
      1               then

                      call maktra ( bin, fft_amplitude ( bin), strong)

                  else

                      call maktra ( bin, fft_amplitude ( bin), weak)

                  end if

              end if

          else

              looking_for_downslope = .true.

          end if

      end do

      call compress

      end
      subroutine smooth ( dynamic_tracking_threshold, faca, alpha, beta,
      1    max_fft_bin_number)

      implicit none

      real            alpha ( 0:*)
      real            beta ( 0:*)
      real            dynamic_tracking_threshold
      real            faca
      integer         max_fft_bin_number

      integer         i
      integer         track

      include         'tracks.inc'
```

```
        do track = 1, number_of_tracks

            i = faca * t_adaptive_amplitude ( track)

            if ( i .lt. 0) then
                i = 0
            else if ( i .gt. 31) then
                i = 31
            end if

            if ( t_dcell ( track) .ne. 0) then
                t_frequency ( track) = t_frequency ( track) +
     1              alpha ( i) * t_dcell ( track)

                if ( t_adaptive_amplitude ( track) .ge.
     1              dynamic_tracking_threshold) then

                    t_frequency_rate ( track) = t_frequency_rate ( track) +
     1                  beta ( i) * t_dcell ( track)
                end if

            end if

            t_frequency (track) = t_frequency ( track) +
     1          t_frequency_rate ( track)

            if (( t_frequency ( track) .gt. max_fft_bin_number) .or.
     1          (t_frequency ( track) .lt. 1.)) call deltra ( track)

        end do

        call compress

        end
        subroutine sort

        implicit none

        integer         track1
        integer         track2

        include         'tracks.inc'

        do track1 = 1, number_of_tracks - 1
            do track2 = track1 + 1, number_of_tracks

                if ( t_frequency ( track1) .gt. t_frequency ( track2)) then

                    call swap_real ( t_adaptive_amplitude ( track1),
     1                  t_adaptive_amplitude ( track2))

                    call swap_integer ( t_age ( track1), t_age ( track2))

                    call swap_real ( t_associated_amplitude ( track1),
     1                  t_associated_amplitude ( track2))

                    call swap_real ( t_associated_cell ( track1),
     1                  t_associated_cell ( track2))
```

```
            call swap_real ( t_dcell ( track1), t_dcell ( track2))

            call swap_logicall ( t_detection_flag ( track1),
     1          t_detection_flag ( track2))

            call swap_real ( t_frequency  ( track1),
     1          t_frequency  ( track2))

            call swap_real ( t_frequency_rate ( track1),
     1          t_frequency_rate ( track2))

            call swap_integer ( t_id ( track1), t_id ( track2))

            call swap_real ( t_log_likelihood ( track1),
     1          t_log_likelihood ( track2))

            call swap_integer ( t_type ( track1), t_type ( track2))


          end if

        end do

      end do

      end
```

```
      subroutine swap_real ( x, y)

      real    x
      real    y

      real    z

      z = x
      x = y
      y = z

      end


      subroutine swap_integer ( x, y)

      integer x
      integer y

      integer z

      z = x
      x = y
      y = z

      end


      subroutine swap_logicall ( x, y)

      logical*1       x
      logical*1       y

      logical*1       z

      z = x
      x = y
      y = z

      end
      subroutine update ( amplitude_smoothing_constant, detection_threshold,
     1    display_threshold, drop_threshold, log_likelihood_max)

      implicit none

      real            amplitude_smoothing_constant
      real            detection_threshold
      real            display_threshold
      real            drop_threshold
      real            log_likelihood_max

      real            log_likelihood_ratio
      real            tll
      real            temp_real
      integer         track
```

```
      include         'tracks.inc'
```

```
      do track = 1, number_of_tracks

          t_log_likelihood ( track) = t_log_likelihood ( track) +
     1        log_likelihood_ratio ( t_type ( track),
     1        t_associated_amplitude ( track))

          t_log_likelihood ( track) =
     1        min ( t_log_likelihood ( track) , log_likelihood_max)


          tll = t_log_likelihood ( track)

          if ( tll .le. drop_threshold) then

              call deltra ( track)

          else if ( tll .lt. display_threshold) then

              t_detection_flag ( track) = .false.
              if ( t_type ( track) .eq. edt)
     1            t_age ( track) = t_age ( track) + 1


          else if ( tll .lt. detection_threshold) then

              if ( t_detection_flag ( track)) then

                  t_age ( track) = t_age ( track) + 1

              else

                  if ( t_type ( track) .eq. edt)
     1                t_age ( track) = t_age ( track) + 1


              end if

          else

              if ( t_type ( track) .eq. edt) then

                  t_detection_flag ( track) = .true.
                  t_age ( track) = t_age ( track) + 1

              else

                  t_type ( track) = edt
                  t_age ( track) = 1
                  t_detection_flag ( track) = .true.

              end if

          end if

      temp_real = t_associated_amplitude ( track) -
     1    t_adaptive_amplitude ( track)
```

```
        t_adaptive_amplitude ( track) = t_adaptive_amplitude ( track) +
        1    temp_real / amplitude_smoothing_constant

        end do

        end
        subroutine writestuff ( k, fft_amplitudes,thr,pct,nop,fnam)
c
c
c       writestuff calculates and displays several track parameters
c
c
c
c
c       declarations
c
c
        implicit none
        real            fft_amplitudes(512),level
        real            thr,pct,dif
        integer*4       fl,fh,nom,noh,nmr,i,j,k,lun,nop
        include         'tracks.inc'
        character*80    fnam
c
c
c       set up output file if k=1
c
c
        if (k.eq.1) then
                lun=1
                open(unit=lun,file=fnam,status='new',carriagecontrol='list')
        endif

c
c
c       calculate average signal+noise level
c
c
        level=0.
        do i=1,512
        level=level+fft_amplitudes(i)
        enddo
        level=level/512.
c
c
c       print header
c
c
C       type 1000, k, number_of_tracks
C
C1000   format ( /'0Track table dump at line ',i3, ' contains ' i3,
C       1    ' tracks.'/)
C
C       type 1005
C
C1005   format ( ' track det   freq    freq    amp     ass     ass     fl'
C       1    '      fh   diff  level'/
C       1    '   #                 rate            freq    amp'/)
```

```
c
        if (number_of_tracks.ge.1) write(lun,891) k
891     format('(',i4)
c
c
c       main loop
c
c
        do 100 i = 1, number_of_tracks
c
c       find fl,fh,diffusion
c
c       fh first
c
        j=t_associated_cell(i)
        nmr=0
        noh=1
        nom=0
776     j=j+1
        if (fft_amplitudes(j).ge.thr*level) then
                noh=noh+1
                nmr=0
        else
                nom=nom+1
                nmr=nmr+1
                if ((nmr.ge.2).and.(pct.ge.(float(noh)/float(noh+nom))))
     x                  go to 777
        endif
        go to 776
777     continue
        j=j-1
        if ((fft_amplitudes(j).lt.thr*level).and.(j.ne.t_associated_cell(i)))
     x          go to 777
        fh=j
c
c       do fl
c
c
        j=t_associated_cell(i)
        nmr=0
        noh=1
        nom=0
778     j=j-1
        if (fft_amplitudes(j).ge.thr*level) then
                noh=noh+1
                nmr=0
        else
                nom=nom+1
                nmr=nmr+1
                if ((nmr.ge.2).and.(pct.ge.(float(noh)/float(noh+nom))))
     x                  go to 779
        endif
        go to 778
779     continue
```

```
        j=j+1
        if ((fft_amplitudes(j).lt.thr*level).and.(j.ne.t_associated_cell(i)))
     x          go to 779
        fl=j
c
c
c       find diffusion
c
c
        dif=0
        do j=fl,fh
        dif=fft_amplitudes(j)+dif
        enddo
        dif=dif/(float(fh-fl+1))


C           type 1001, i,
C       1       t_detection_flag ( i),
C       1       t_frequency  ( i),
C       1       t_frequency_rate(i),
C       1       t_adaptive_amplitude (i),
C       1       t_associated_cell ( i),
C       1       t_associated_amplitude ( i),
C       1       fl,
C       1       fh,
C       1       dif,
C       1       level
C
C1001   format ( i4,15,f8.2,f8.2,f7.2,f8.0,f8.2,i7,i7,f7.2,f7.2)
c
c
c
c       write stuff to output file
c

        write(lun,889) t_detection_flag ( i),
        1       t_frequency  ( i),
        1       t_frequency_rate(i),
        1       t_adaptive_amplitude (i),
        1       t_associated_cell ( i),
        1       t_associated_amplitude ( i),
        1       fl,
        1       fh,
        1       dif,
        1       level
889     format ( '(',15,f8.2,f8.2,f7.2,f8.0,f8.2,i7,i7,f7.2,f7.2,')')

c
c
c
c
c
100     continue
c
        if (number_of_tracks.ge.1) write(lun,890)
890     format(')')
c
c
```

```
c
c       close output file
c
        if (j.eq.nop) close(unit=lun,status='keep')
c
c
c
        end
```

```
c       JHJLIB.INC

        integer*4       ap_max_address
        parameter       ( ap_max_address = 16383 )

        integer*4       array_processor
        parameter       ( array_processor = 2 )

        integer*4       askflt_l
        parameter       ( askflt_l = 1 )

        integer*4       askflt_luv
        parameter       ( askflt_luv = 7 )

        integer*4       askflt_lv
        parameter       ( askflt_lv = 5 )

        integer*4       askflt_lu
        parameter       ( askflt_lu = 3 )

        integer*4       askflt_nodefault
        parameter       ( askflt_nodefault = 0 )

        integer*4       askflt_u
        parameter       ( askflt_u = 2 )

        integer*4       askflt_uv
        parameter       ( askflt_uv = 6 )

        integer*4       askflt_v
        parameter       ( askflt_v = 4 )

        integer*4       askint_l
        parameter       ( askint_l = 1 )

        integer*4       askint_luv
        parameter       ( askint_luv = 7 )

        integer*4       askint_lv
        parameter       ( askint_lv = 5 )

        integer*4       askint_lu
        parameter       ( askint_lu = 3 )

        integer*4       askint_nodefault
        parameter       ( askint_nodefault = 0 )

        integer*4       askint_u
        parameter       ( askint_u = 2 )

        integer*4       askint_uv
        parameter       ( askint_uv = 6 )

        integer*4       askint_v
        parameter       ( askint_v = 4 )

        integer*4       complex_floating
        parameter       ( complex_floating = 3 )
```

```
      integer*4     complex_integer
      parameter     ( complex_integer = 4 )

      integer*4     memory
      parameter     ( memory = 1 )

      integer*4     read_access
      parameter     ( read_access = 1 )

      integer*4     real_floating
      parameter     ( real_floating = 1 )

      integer*4     real_integer
      parameter     ( real_integer = 2 )

      integer*4     write_access
      parameter     ( write_access = 2 )

      real          amplitude_smoothing_constant
      real          association_gate_min ( 3)
      real          association_gate_max
      real          association_gate_slope ( 3)
      real          association_threshold
      real          detection_threshold
      real          display_threshold
      real          drop_threshold
      real          dynamic_tracking_threshold
      real          faca
      real          log_likelihood_max
      integer       max_fft_bin_number
      real          merge_gate
      real          new_track_threshold ( 3)
      real          rate_gate

      common / param /
      1     amplitude_smoothing_constant,
      1     association_gate_min,
      1     association_gate_max,
      1     association_gate_slope,
      1     association_threshold,
      1     detection_threshold,
      1     display_threshold,
      1     drop_threshold,
      1     dynamic_tracking_threshold,
      1     faca,
      1     log_likelihood_max,
      1     max_fft_bin_number,
      1     merge_gate,
      1     new_track_threshold,
      1     rate_gate

c     tracks.inc

      integer       edt
      parameter     ( edt = 3)
      integer       max_tracks
      parameter     ( max_tracks = 400)
```

```
        integer         strong
        parameter       ( strong = 2)
        integer         weak
        parameter       ( weak = 1)

        integer         free_entry
        integer         number_of_tracks
        real            t_adaptive_amplitude ( max_tracks)
        integer         t_age ( max_tracks)
        real            t_associated_amplitude ( max_tracks)
        real            t_associated_cell ( max_tracks)
        real            t_dcell ( max_tracks)
        logical*1       t_detection_flag ( max_tracks)
        real            t_frequency  ( max_tracks)
        real            t_frequency_rate ( max_tracks)
        integer         t_id ( max_tracks)
        real            t_log_likelihood ( max_tracks)
        integer         t_type ( max_tracks)

        common / tracks/

        1    free_entry,
        1    number_of_tracks,
        1    t_adaptive_amplitude,
        1    t_age,
        1    t_associated_amplitude,
        1    t_associated_cell,
        1    t_dcell,
        1    t_detection_flag,
        1    t_frequency ,
        1    t_frequency_rate,
        1    t_id,
        1    t_log_likelihood,
        1    t_type
```

```
8.0         amplitude_smoothing_constant
3.0         association_gate_min
3.0         association_gate_max
0.0         association_gate_slope (weak)
0.5         association_gate_slope (strong)
0.5         association_gate_slope (edt)
2.25        association_threshold
0.5         old_track_pd
1e-4        old_track_pfa
0.0         delta_display_threshold_from_detection_threshold
2.25        dynamic_tracking_threshold
4.0         faca
10.017      log_likelihood_max
512         max_fft_bin_number
5.0         merge_gate
0.5         rate_gate
0.03        new_track_pfa
```

END

DTIC

5-86